Funambol Mobile Open Source

Sync your e-mail and other data with mobile devices using Funambol 7.1

Stefano Fornari

PACKT PUBLISHING

BIRMINGHAM - MUMBAI

Funambol Mobile Open Source

First published: December 2009

Production Reference: 1111209

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-847191-54-0

www.packtpub.com

Cover Image by Vinayak Chittar (vinayak.chittar@gmail.com)

Credits

Author
Stefano Fornari

Reviewer
Alberto Falossi

Acquisition Editor
Sarah Cullington

Development Editor
Swapna Verlekar

Technical Editors
Ajay Chamkeri
Gaurav Datar

Indexer
Rekha Nair

Editorial Team Leader
Gagandeep Singh

Project Team Leader
Lata Basantani

Project Coordinator
Srimoyee Ghoshal

Proofreader
Lesley Harrison

Graphics
Nilesh R Mohite

Production Coordinator
Shantanu Zagade

Cover Work
Shantanu Zagade

About the Author

Stefano Fornari is cofounder and CTO of Funambol Inc., the leading provider of open source mobile cloud sync and push e-mail solutions for billions of phones. Stefano had several years of software development experience before starting the Funambol open source project in 2003. He is one of the main contributor of the project and the project manager. Stefano is also in charge of the engineering team of Funambol. Prior to his development work on Funambol, Stefano was chief architect at Stigma Online, where he played a key role in the development of the company's flagship portal product, SolWeb Intra. He has also held positions at Compaq, where he was an advisor on wireless technologies in PDAs, and at Art Technology Group (ATG) as a consultant. He holds an M.S. in Computer Science.

Acknowledgement

This is the hardest part of the book. It is never easy to make sure you do not forget anyone, but I'll try. A book about Funambol is not only about a software product, it is about a venture started quite a while ago. Funambol would not exist if it was not for the people who made it. Therefore, a big thank first of all to those who started this venture with me, Fabrizio Capobianco and Alberto Onetti.

Particular thanks go to Giulia Zanchi and Hal Steger for their great reviews and suggestions and improvements; to Mirco Porcari, who contributed substantially to Chapter 7; to Federico Casali for his feedback; to Marco Magistrali, Ivano Brogonzoli, Matteo Vitolo for their help with Funambol clients.

Lastly a big thank to all Funambolists, without your effort, dedication and ability to walk on a tight rope, I would have never got a topic to write about.

About the Reviewer

Alberto Falossi is an ICT consultant, specialized in New Media and Enterprise 2.0. He helps companies reengineer their business with IT technologies. He is Contract Professor at the University of Pisa, Italy. Alberto is a Web Entrepreneur, and founded some popular startups such as `www.kapipal.com` and `www.opinionauto.com`. You can reach him through his blog at `www.albertofalossi.com`.

Dedicated To
Simona, Arianna, Laura

Table of Contents

Preface

Why a book on mobile open source sync and push e-mail? Because mobile applications and messaging are the next big thing. Over 750 million mobile devices are sold every year. The entire population of the planet will have a mobile device (in many cases, more than one) in a few years. The diffusion of mobile devices is going to surpass, by every measurement, that of personal computers. If the Internet enabled an explosion of web applications, then the new era of wireless will create incredible demand for mobile applications. One of the biggest demand is already clear nowadays. People want to keep their network of contacts always with them even when they are not at their desktop. The need for sharing contacts and messages on a wide range of devices (in particular mobile) and with a wide range of applications has become crucial. People want to keep communicating using any means of communication they have used already—voice, e-mail, Facebook, blogs, and so on.

How this can be achieved in a landscape of so divergent applications and devices? With open standard protocols and open source implementations!

This book helps you gradually learn how to provide a full featured PIM synchronization and push e-mail service. It explains in details how to set up a Funambol service that provides mobile and desktop users contacts, along with calendar synchronization and push e-mail. It starts from a simple scenario of a personal address book backup scenario to mobilize enterprise data in a realistic enterprise environment. The book is also a great starting point for those who aim to extend Funambol in a truly open source spirit.

Another reason why I decided to write this book is because I see this as a valuable contribution to the Funambol community. As I like to say when I hire Funambol developers, working for Funambol is not only about working for a company; it means being part of a community of users, developers, and projects that make a few lines of code something alive and vital, something that changes and grows every day with the contribute of everyone.

I hope you will enjoy reading the following pages and learning about Funambol, mobile sync, and push e-mail. I certainly enjoyed writing this book and I hope to see you as part of the Funambol community soon.

Structure of the book

The book is structured in two parts.

In the first part, which contains Chapter 1 to 8, we will help Maria, the IT manager of a medium size enterprise, to deploy Funambol in her organization. We will first show them what Funambol is about (Chapter 1) and of which systems it is built of (Chapter 2). The next chapters focus on the administration functionality and tasks Maria may need for an efficient operation of Funambol, such as: administering users and devices, managing Funambol push e-mail and PIM synchronization (Chapter 4, 5, and 6). The following chapter (Chapter 7) shows how many different devices and applications can be easily used with Funambol to cover many scenarios that can be found in the real life. Finally, the last

chapter (Chapter 8), tells Maria how Funambol can be extended with existing connectors to integrate with existing back ends like a CRM system.

The second part of the book is instead intended for developers who want to extend Funambol with integrations to different back ends. First, an introduction to the SyncML synchronization protocol is given (Chapter 9), so to give a common background developers can refer to; finally, a detailed description and tutorial about how to write Funambol connectors is given in the last chapter (Chapter 10).

What this book covers

PART 1 – Deploying Funambol

Chapter 1: *The First Sync* shows how to install the server on both Linux and Windows and how to start/stop Funambol services. The reader will be guided to complete the first sync so as to make sure everything works as expected.

Chapter 2: *Synchronizing Funambol* shows how Funambol allows a user to sync their data and get notified of new e-mails or other personal data changes. The chapter covers what the architecture looks like from a system perspective, which components are involved and their responsibilities, and how Funambol push works.

Chapter 3: *Mastering Data Synchronization* tells how to comfortably configure and manage the Data Synchronization service. It will also provide general configuration concepts, acquiring the background and information needed to go in further detail in the following chapters.

Chapter 4: *Administering Users and Devices* will explain how to manage users and devices through the administration tool.

Chapter 5: *Funambol E-mail* discusses how to comfortably configure and manage the Funambol E-mail Connector.

Chapter 6: *Funambol PIM Data Push* shows how to comfortably work with Funambol PIM push.

Chapter 7: *Synchronizing Devices and Desktops* helps configure Funambol and other SyncML clients and how to synchronize devices from end to end.

Chapter 8: *Making the Most of Funambol: Community and its Projects* helps the reader learn about the Funambol community, how this can help a Funambol administrators and users, and which tools are available to users and developers. The reader will also learn how she can help the Funambol product or other users. In addition, as an example of how the community contributes to add value to the product, the reader will be presented a community project to integrate one of the most known CRM called SugarCRM.

PART 2 – Extending Funambol

Chapter 9: *Introduction to SyncML*, discusses in more detail, the SyncML protocol used by Funambol to synchronize personal data and e-mail.

Chapter 10: *Extending the Funambol Data Synchronization Service* helps the reader learn how to develop a custom connector to access an external datasource. It will go through a simple example in the form of a tutorial.

What you need for this book

To start this book you do not need any particular software or tool. The book takes the reader through all steps needed to download, install, and use the Funambol mobile open source software. Some system administration background is recommended as the book is about deploying a PIM synchronization and mobile e-mail solution. In the second part of this book, which covers how to extend Funambol, some Java programming background is assumed.

Who this book is for

If you are looking forward to installing and getting started with Funambol, this book is for you. You need to be technically minded and confident with a bit of code tweaking, but not a developer.

General server administration skills are assumed and familiarity with Java will be a benefit in places.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "Standard options of a JVM can be set in the `JAVA_OPTS` environment variable before starting Funambol."

A block of code will be set as follows:

```
DEBUG [main]: Message 1
   WARN  [main]: Message 2
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items will be shown in bold:

```
<void method="setProperty">
    <string>username</string>
    <string>syncuser</string>
   </void>
```

Any command-line input or output is written as follows:

```
$ bin/admin-passwd
Changing password for admin
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "Maria can access the loggers available in the server by expanding the **Server settings | Logging** node ".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an e-mail to `feedback@packtpub.com`, and mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or e-mail `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code for the book

Visit `http://www.packtpub.com/files/code/1540_Code.zip` to directly download the example code.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration, and help us to improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **let us know** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata added to any list of existing errata. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

Part 1

Deploying Funambol

The First Sync

Synchronizing Funambol

Mastering Data Synchronization

Administering Users and Devices

Funambol E-mail

Funambol PIM Data Push

Synchronizing Devices and Desktops

Making the Most of Funambol: Community and its Projects

1
The First Sync

Maria uses her mobile phone to keep in touch with friends and colleagues. She is a busy person and likes to organize her time using her personal computer's calendar. Maria is an open source fan, so whenever possible, she chooses open source software such as Mozilla Thunderbird, which she uses for both e-mail and calendaring. It would be really useful for her to have her address book and agenda when she is not at her desk. She would like to be able to update her contact list and take appointments when traveling, and have these changes automatically saved to her personal computer, so when she gets home she can resume work without further action.

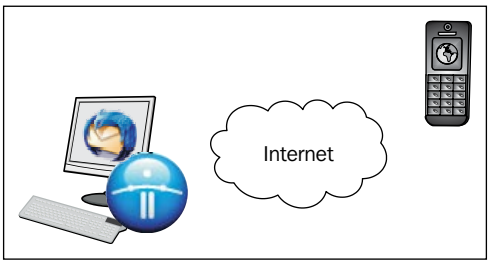
Funambol is designed for deployments that support millions of users; however, the basic setup package can be installed on a personal computer for personal use. In this chapter, we will learn how to install Funambol to keep our mobile phones, computer address books, and calendars in sync.

In this chapter, we will follow Maria as she installs and starts using Funambol to solve her synchronization issues.

Overview

As illustrated in the following figure, all Maria needs is a computer connected to the Internet on which Mozilla Thunderbird, her preferred e-mail client, is installed and on which Funambol will also be installed. Thanks to SyncML, the only other device needed is a SyncML-capable mobile phone.

SyncML is a widely adopted synchronization protocol that can work in both the cases—over the air or wired. The vast majority of modern mobile phones have SyncML onboard, usually accessed through an Internet connection. This feature is usually called remote synchronization or simply synchronization and allows to synchronize the phone address book, calendar, notes and sometimes even SMS. With these phones, users do not need to download or install any additional software; they will just have to configure the SyncML synchronization feature to connect to a proper service. Maria will see all these aspects in detail in the next chapters.



After setting up Funambol, when Maria is working at her desk, any changes to her contacts or calendar will be sent by Thunderbird to the Funambol Server. When she is on the road, the changes made on the phone will be synchronized to the Funambol Server, and from there to Thunderbird.

We will assume that Maria is working with the following systems:

- Windows XP/Vista personal computer
- Mozilla Thunderbird 2.0
- A SyncML-capable phone

Working with Maria's personal computer

The Funambol Server package comes in two OS-specific versions, one for Microsoft Windows and one for Linux distributions. This section details the software and hardware requirements for a successful installation on Microsoft Windows.

Minimum system requirements

Maria's personal computer must meet the following minimum system requirements:

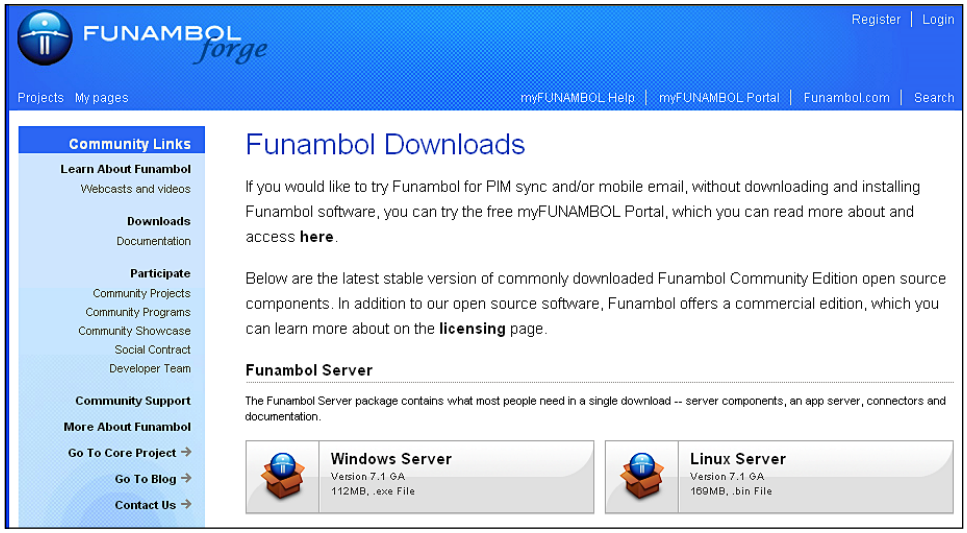
- Pentium 4 CPU, running at 1.8GHz
- Windows XP Professional or Windows Vista
- 250 MB of free disk space
- 768 MB RAM

Obtaining and installing the Funambol Server package

A personal installation of Funambol can be achieved from the Funambol Server package that contains everything needed to run all the Funambol services on a single system, potentially a Personal Computer. Note that this type of installation is intended for personal use and evaluation only. The next chapters describe how to address more complex needs and scenarios.

The book covers Funambol v7.1, which is the version available at the time of writing. However, everything described in the book will work smoothly with the upcoming Funambol v8.

The first thing that Maria must do is download the Funambol Server package for Windows from `http://www.forge.funambol.org/download` as shown in the following screenshot:



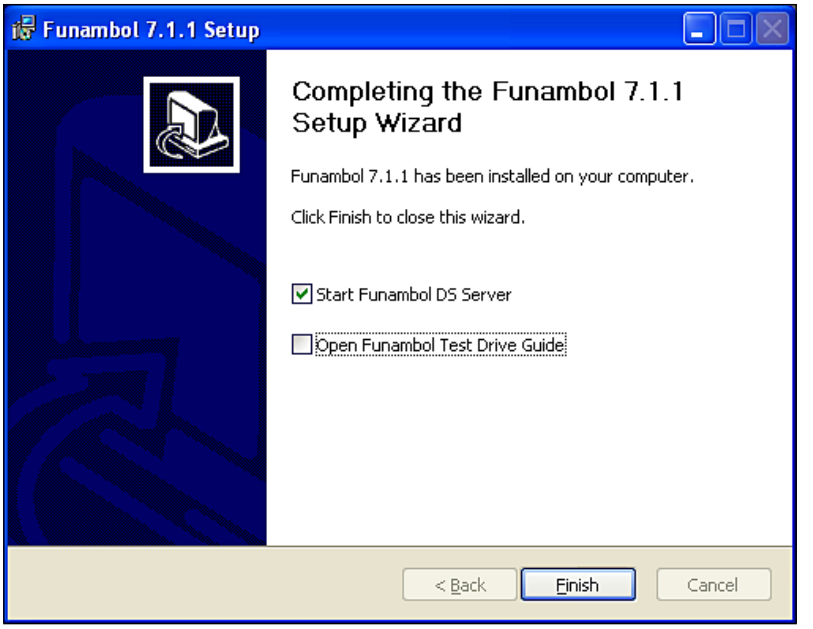
The file is called `funambol-<version>.exe`, for example, `funambol-7.1.1.exe`. Once it has been downloaded, Maria can double-click it to start the setup wizard, as shown in the next screenshot:

After accepting the open source license and all of the default options suggested by the wizard, the software installation will complete in a few seconds.

The setup program loads the following key components onto the computer:

- Funambol Data Synchronization Service
- Java Runtime Environment, 1.5.x
- Hypersonic database
- Funambol Administration Tool
- Software accessories/emulators for testing

At the end of installation, the setup wizard will ask Maria if she wants to start the server and read the Test Drive Guide. Let's start the server and ignore the Test Drive Guide by unchecking the **Open Funambol Test Drive Guide** option as shown in the following screenshot:



SyncML is transported on an Internet connection over the HTTP protocol. This means the server is fundamentally an HTTP server, even if its primary goal is not to serve web pages. By default, the Funambol Server starts listening on port 8080.

Maria can check whether the server has started and is running, by looking at the system tray that displays the Funambol status icon as highlighted in the following screenshot:

Preparing the e-mail client

In this section, we will see how Maria can configure Mozilla Thunderbird to get the most out of Funambol. The Funambol Sync Client for Thunderbird can synchronize both address books and calendars. Out of the box, Thunderbird 2.x does not come with a calendar application, so Maria will need to install a Thunderbird extension called Lightning that can be found in the Thunderbird add-ons directory at `http://addons.mozilla.org/thunderbird`. She can download the extension by searching for "Lightning" on the website and then clicking **Download Now** to save the file `lightning-<version>-tb-win.xpi` on her computer.

The extension can be installed from Thunderbird by selecting **Tools | Add-ons** and then clicking **Install** in the window that pops up. Maria then needs to select the Lightning extension setup file downloaded previously and click **Install Now**.

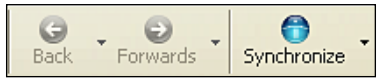
After a few seconds, the setup window will ask to restart Thunderbird to complete the installation of the extension. When Thunderbird restarts, Maria will find the **Events** panel on the right pane of the e-mail client window and a **Today Pane** button in the button bar.

The Funambol Sync Client can be found in the Mozilla Thunderbird Add-ons directory by searching for "Funambol". The Funambol Mozilla Sync Client extension is a Funambol community project, developed by an external contributor (see `https://mozilla-plugin.forge.funambol.org`). Maria should download the Windows version and save it on the hard drive. The name of the file will be in the form `funambol_mozilla_sync_client-<version>-tb+sb-win.xpi`.

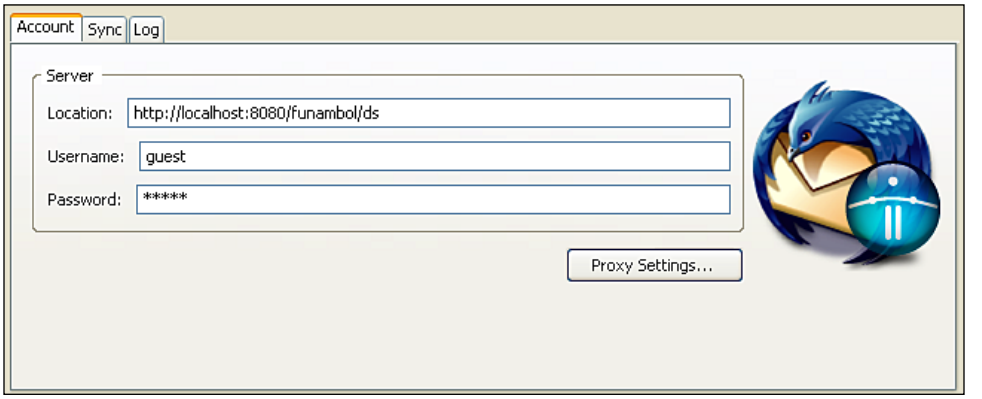
Community projects are those projects that may receive contributions from Funambol developers, but they are started and/or maintained by people who may not have any relationship with the Funambol company Funambol core developers.

At the time of writing, the most recent release of the Funambol Mozilla Sync Client is version 0.9. This is still flagged as experimental, and so you must register and log in to the Mozilla website before downloading it. This is a normal procedure for Mozilla extensions—they are tested for a while as experimental, before becoming official.

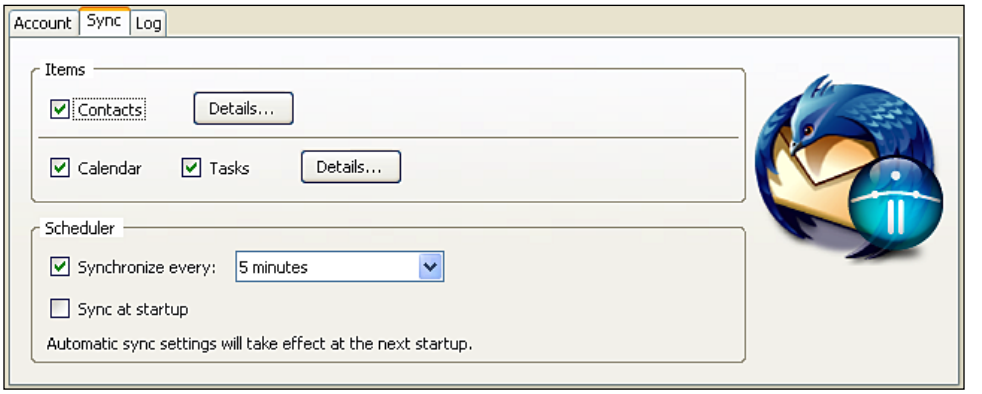
The installation of the Funambol Mozilla Sync Client is similar to the installation of Lightning. It can be done from Thunderbird by going to the **Tools | Add-ons** menu, then selecting the previously downloaded file and installing it. Again, Maria will be requested to restart Thunderbird. Once restarted, the Thunderbird button bar will have an additional button to launch the Funambol add-on as illustrated in the following screenshot:



At this point, Maria needs to tell the Funambol Mozilla Sync Client to connect to the Funambol Server installed on her desktop computer. She can do it by selecting **Options** from the Funambol icon drop-down menu and then configuring the **Account** and **Sync** settings. Following is the screenshot showing configuration of **Account** settings (note that the **Password** is guest):



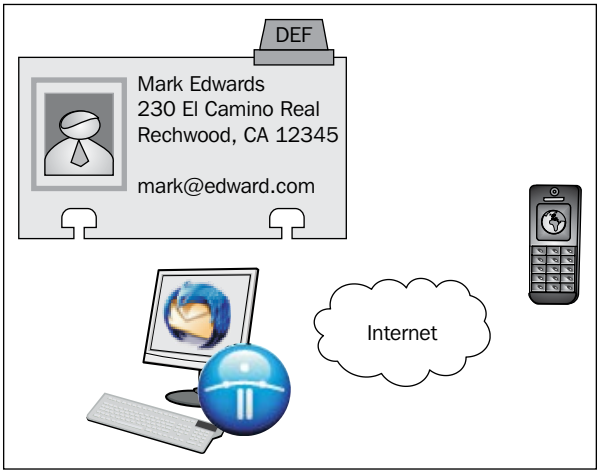
The next is the screenshot showing **Sync** setting configuration:



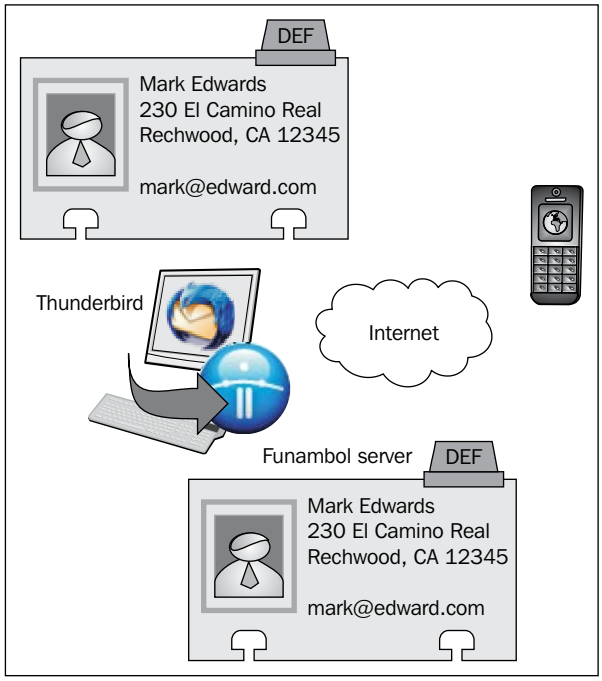
Once the settings have been configured, Maria can save the changes by clicking **OK**.

Before starting the first synchronization, Maria should enter some contacts in Thunderbird's Personal address book and some appointments in Thunderbird's Calendar. To synchronize these, she clicks the Funambol icon. The synchronization takes place in a few seconds and the Funambol Thunderbird Sync Client window will display the result of the synchronization.

Consider that Maria has added a card in Thunderbird for her friend **Mark Edwards** as shown in the following image:



After the synchronization, the status will be as represented in the following illustration:



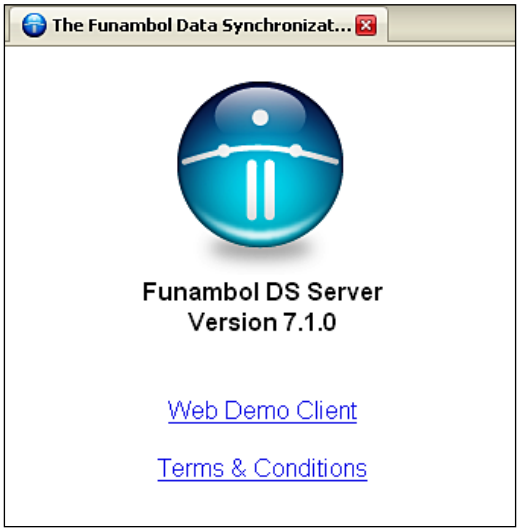
Now Mark Edwards is stored in both the Funambol Server and Maria's address book.

Preparing the computer for Internet access

The first synchronization was done by connecting the Funambol Mozilla Sync Client to the Funambol Server installed on the local PC. A mobile phone cannot usually do the same because it is on a different network, usually the GPRS/UMTS network provided by the mobile operator, and can only reach the desktop server through the Internet. Therefore, the personal computer must be connected to the Internet with a public IP address (and optionally a host name) that can be accessed by the outside world.

The steps used to configure your system so that it is externally accessible are beyond the scope of the book. If you need more details and help, please contact your service provider.

Let's assume the IP address of Maria's personal computer is routable and is 87.1.21.205. To check that the server is up and running, she can point the browser on her mobile phone to the URL `http://87.1.21.205:8080/funambol`. If the server is reachable, the following image is displayed:



Preparing the mobile phone

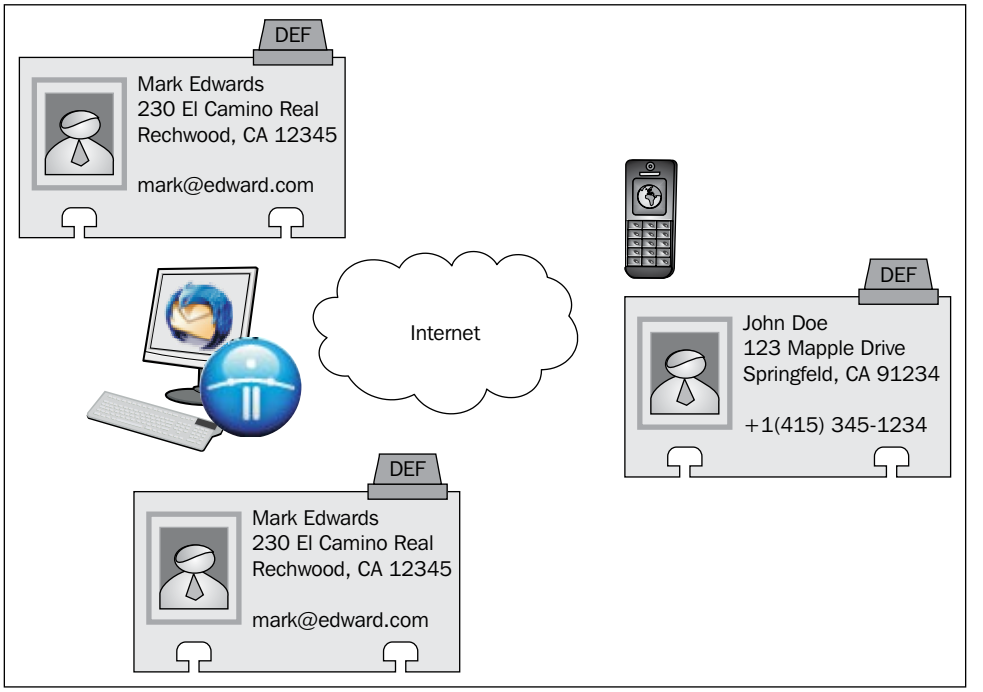
The Funambol Server uses SyncML as its synchronization protocol, which is used by most mobile phones today. You can find a list of phones that support SyncML at `http://www.funambol.com/solutions/devices.php`.

To synchronize the address book and calendar on her mobile phone, a Nokia 6650, Maria will first need to configure the phone using the following steps (for a more general description of the configuration of different devices, please refer to Chapter 7):

1. Go to **Menu** and choose **Tools.**
2. Select **Sync** (on some devices this option is under **Menu** | **Settings** | **Connectivity**).
3. Select **Options** | **New sync profile.**
4. Set the Sync profile name to **Funambol** and press **OK.**
5. Select **Applications.**
6. Select **Contacts.**
7. Set **Include in sync** to Yes.
8. Set **Remote database** to card.
9. Press **Back.**
10. Select **Calendar.**
11. Set **Include in sync** to Yes.
12. Set **Remote database** to cal.
13. Press **Back** twice.
14. Select **Connection settings.**
15. Set the **Server Version** to 1.2.
16. Set **Data bearer** to Internet.
17. To set an **Access point**, enter a working Internet connection (if you don't have one, contact your mobile provider).
18. Set the **Host address** (in the previous example, this was `http://87.1.21.205:8080/funambol/ds`).
19. Set **Port**: 8080.
20. Set **User name**: guest.
21. Set **Password**: guest.
22. Press **Back** to save the changes.

Completing the first synchronization

Following the example described earlier, let's assume Maria's phone address book contains the phone number of John Doe.



Maria can start the synchronization of her mobile phone by following these steps:

1. Go to **Menu** | **Tools**.
2. Select **Sync** (on some devices this option can be accessed as **Menu** | **Settings** | **Connectivity**).
3. Select the Funambol profile. Then select **Options** | **Synchronize** to start sync.

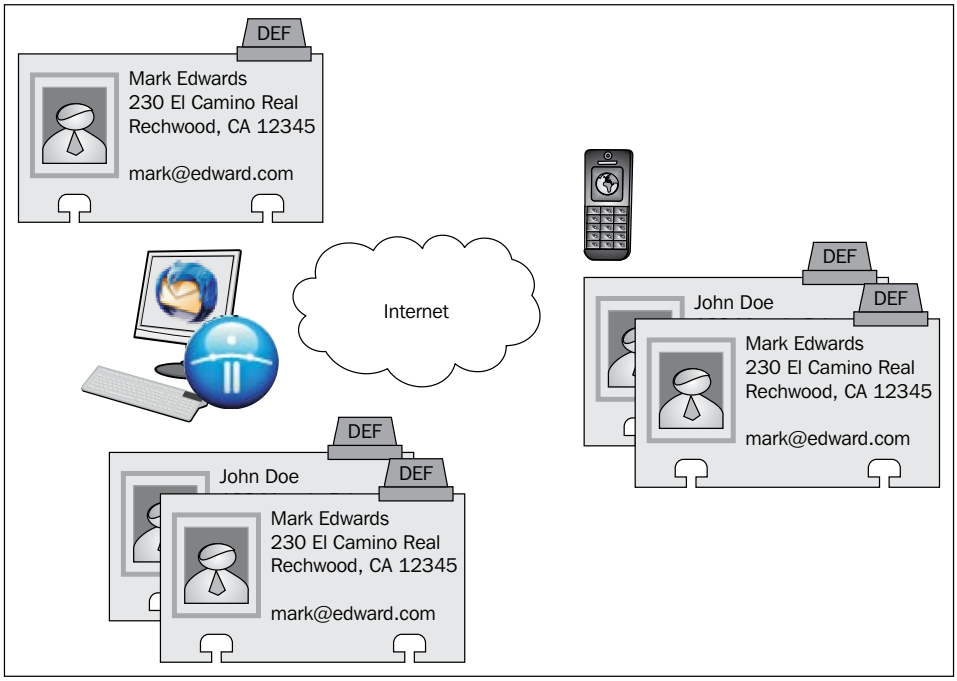
If your mobile asks what data to sync, select **Contacts** and then **Calendar**.

If your phone returns an error, this may be due to network coverage or remote server reachability issues; often restarting the sync solves the problem.

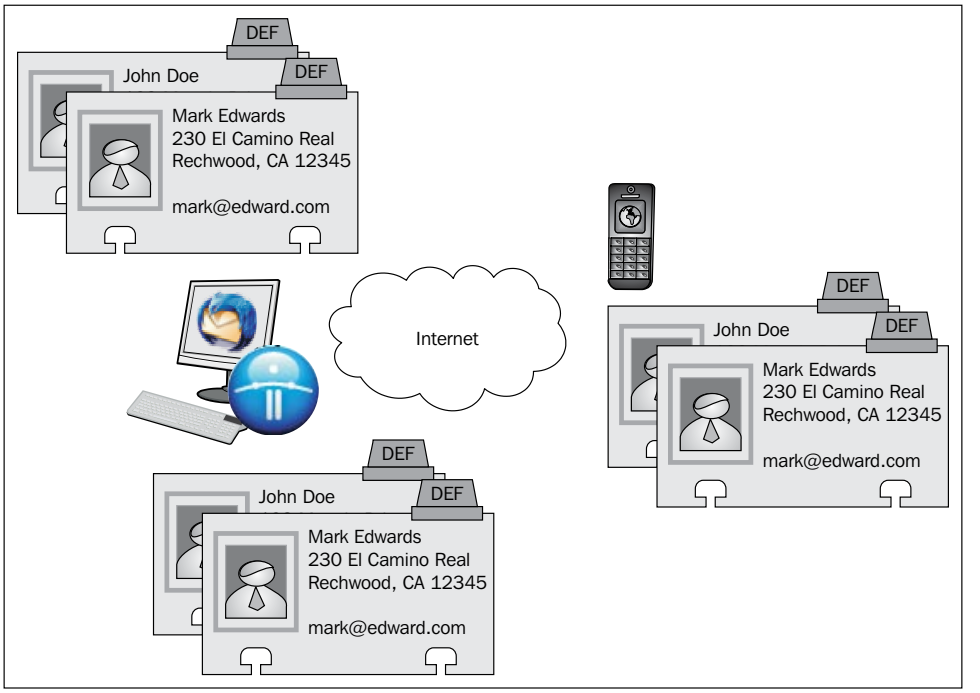
Make sure that the access point selected in the previous steps is under a flat billing data plan, as data traffic can be expensive.

Once the synchronization ends successfully, the server contains a merged version of Maria's address books and calendars stored on her phone and Mozilla Thunderbird.

The new status of the example will be:



As seen in the previous figure, Thunderbird is now out of sync because it is missing John Doe. Therefore, to complete the first synchronization scenario on her desktop, Maria needs to synchronize again using the Funambol Mozilla Sync Client. This can be done by clicking on the Funambol icon on the Thunderbird menu bar. After a few seconds, depending on how many contacts and appointments need to be transferred from the server, the job is done, as illustrated in the next figure:



Maria can now open the address book and have a look at the calendar to check that all contacts and appointments contained in the phone have been copied to Thunderbird as well.

Note that if the same contact is present on the phone and on Thunderbird's address book, then the address book entry will not be duplicated, but instead the information coming from the two sources

will be merged in one single contact. For example, if Maria has her husband's mobile phone number on the phone and his e-mail address in Thunderbird, the resulting contact will contain both the mobile phone number and the e-mail address.

Installing Funambol on Linux

In the previous sections we saw how Maria installed Funambol on her Microsoft Windows system; it is also possible to install Funambol on a Linux desktop computer. This installation requires the use of a terminal window and a command line shell. Have the Funambol Server package version number available (as indicated in the downloaded package filename) for use in this procedure. You do not need to log in as root to complete this installation; any administrator account can perform this task.

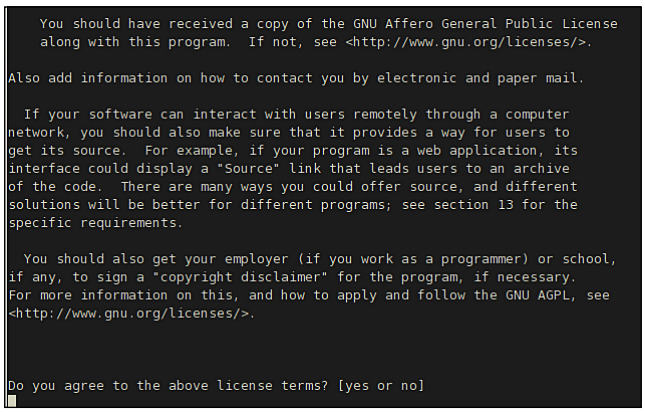
To proceed with the installation, follow these steps:

1. Open a terminal window and type the following command:

   ```
   sh funambol-<version number>.bin
   ```

   press *Enter* to proceed.

   When the license agreement appears, read the text; to accept the terms of the agreement, type **yes** at the prompt and press *Enter*.



2. Specify a directory in which to install the bundled software; by default, Funambol will be installed to the following directory:

   `/opt/Funambol`

   In this book, we will refer to this directory as `<FUNAMBOL_HOME>`.

3. At the end of the installation, you will be prompted to start the server. At the prompt, type **yes** and press *Enter*.
4. If you prefer to delay the startup of the server, you can always change to the `<FUNAMBOL_HOME>` directory and run the following command at a later time:

   ```
   sh bin/funambol.sh start
   ```

This concludes the installation.

Verifying the Funambol Server startup

After installing and starting the Data Synchronization Service, you can verify that it is up and running using one of these methods:

- Run a `ps` command; when the results appear, use the `grep` command to search for the string "funambol". If a listing is found, the server is in operation.
- Start a web browser and link to this URL: `http://localhost:8080/funambol`.

  If the server is up and running, the Funambol Data Synchronization Service test page (illustrated earlier) should appear .

Once the server is installed, the rest of the procedure (described earlier in the chapter) for a Microsoft Windows system is still valid. Just make sure that Thunderbird 2.0 for Linux is installed and then download and install the Linux version of the Funambol Mozilla Sync Client.

Summary

In this chapter, we learned how to set up a personal computer and a mobile phone to keep the address book and calendar on Mozilla Thunderbird 2.0 in sync with the address book and calendar on the mobile phone. The steps followed to achieve this were:

- Installing Funambol on the personal computer that is set up to be accessed from the Internet
- Installing the Lightning extension and the Funambol Mozilla Sync Client into Mozilla Thunderbird
- Configuring the mobile device
- Synchronizing Thunderbird with the Funambol Server, then the mobile phone with the Funambol Server, and then again synchronizing the Thunderbird

We have also learned how to install the Funambol Server on a Linux machine in addition to a Microsoft Windows system.

In the next chapter, we will learn in detail how Funambol makes all of this function, which components are involved, and the functions they perform.

2
Deploying Funambol

When Maria sees how easy it is to have the same personal data on her desktop and her phone, she is amazed. However, it is not enough to surprise other users. After all, you have an address book to call your friends and an address book to send them an e-mail. So, it should not be that difficult to merge them and make sure you always have the same data in all your address books, even on your iPod!

It is not the most known feature, but the iPod comes with an address book and a calendar application on it.

The technology that lies under the surface, however, is not trivial, considering the number of systems, equipment, and actors involved. Typically, users such as Maria see only a part of the whole picture as they are interested only in their own data.

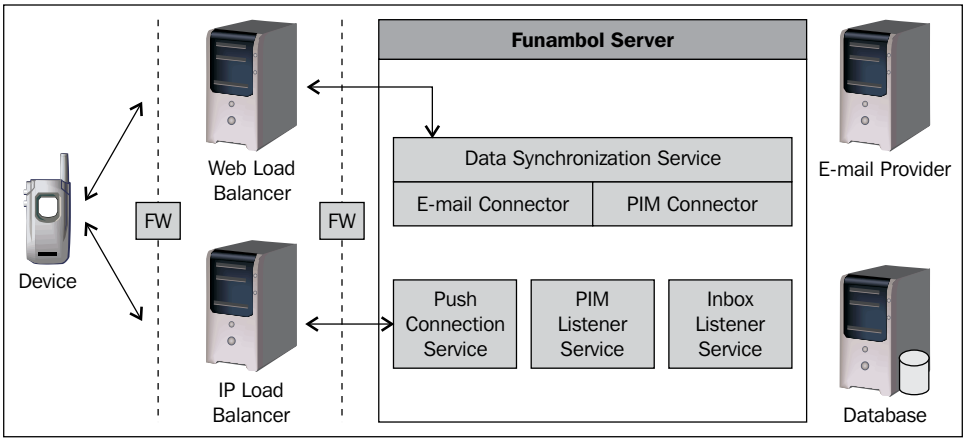
However, for a moment, if we think about the need to support many users at the same time, each one having different devices and datasources, potentially increasing every day ("I bought a new camera, can I sync my pictures so that I always have a backup somewhere?"), the scenario becomes more challenging. How about, for example, integrating the address book that your company's CRM system uses to store customer contacts?

Funambol addresses this challenge. As a solution, it works out of the box, but it can also be extended to meet almost any data synchronization need. Thanks to a powerful and flexible architecture that can scale from a few users or even personal use to a much larger user base.

What we have not yet said about Maria is that she is the IT manager of the company where she works. She is in charge of extending her personal experience with a Funambol deployment to everyone in her company. To do so, she needs to understand the system architecture of a Funambol deployment that can scale to many users. This chapter provides a common background to understand all of the pieces that make up Funambol, which role they play, and how they interact with each other to reach the final goal—to make data synchronization seamless and easy.

Funambol architecture

A full Funambol deployment's system architecture consists of the components illustrated in the following figure:



Note that the Funambol Server setup package installs all services at once. In other words, Maria will not need to choose the proper combination of several installation choices. However, in a complex deployment, not all services will be running at the same time on the same machine. For example, perhaps the Data Synchronization Service will be hosted on one or two nodes and the Listener Services on other nodes. These possibilities are discussed in the next chapters, which describe in detail how to configure and operate each service.

In this chapter, we are going to describe the roles and responsibilities of each component in the Funambol architecture, as illustrated in the previous figure so that Maria can plan to deploy the system in her IT department appropriately.

Device

A device can be any physical device or client software application that can communicate with the Funambol Server using the SyncML protocol. Examples of such devices are:

- Mobile phones with a native SyncML client
- Mobile phones or devices running Funambol Sync Clients (Windows Mobile, iPhone, BlackBerry, JavaME)
- Desktop applications running a Funambol Sync Client (such as Funambol Thunderbird Sync Client, Funambol Outlook Sync Client, and Funambol iPod Sync Client)

Devices are the main interface through which users access Funambol. Responsibilities of the device/client software include:

- Providing the user interface
- Initiating the communication with the server
- Hosting the local data (PIM, e-mail, and so on)
- Collecting/detecting the change log

The term **change log** refers to the recording of what has changed in a datasource since a point in time (for example, the last synchronization). There are many techniques to achieve such result and each device software platform provides its own implementation of the change log.

Funambol Server

The Funambol Server is the core of the Funambol push e-mail service and of PIM synchronization. It includes several components that are described in the following sections.

Data Synchronization Service

The Data Synchronization Service provides synchronization functions and communicates directly with devices using SyncML. Its main responsibilities are:

- Hosting the synchronization engine
- Accepting and serving synchronization requests
- Handling low level device information
- Synclet technology
- Providing an interface to external datasources
- Providing a remote administration interface
- Providing connection-less push

Notes:

The Data Synchronization Service is the only service that the Funambol Server must execute.

Synclet technology is Funambol-specific functionality to preprocess and postprocess SyncML requests, before and after the message is handled by the synchronization engine. Maria or any end user need not deal with synclets directly, they will be working behind the scenes.

Connection-less push is another push mechanism implemented by Funambol that does not require a constant connection between the client and the server. For a better description, see section *Funambol push*.

E-mail Connector

The E-mail Connector plays a key role in the Funambol architecture. It adds support for push e-mail and e-mail synchronization to the Funambol platform. It allows the server to communicate with the user e-mail providers.

For example, if Maria wants to receive e-mails from her Gmail account, the E-mail Connector will connect to Gmail when synchronizing Maria's phone. If one of Maria's colleagues wants to receive her work e-mails on her mobile phone, the E-mail Connector can connect to the company's IMAP/POP server.

The E-mail Connector is deployed with the synchronization engine and has the following responsibilities:

- Determine which messages to download to the client
- Using various techniques to enable the user to filter unwanted e-mails (for example, retrieve and sync headers only)
- Processing e-mails
- Sending outgoing messages

For more information on the life cycle of a message, please see section *Funambol push*.

Inbox Listener Service

To provide push e-mail, it is necessary to detect new e-mails in a user's inbox. This is achieved by the Inbox Listener Service, a separate process with the following responsibilities:

- Creating and maintaining the user inbox cache
- Polling the user inbox regularly to check for new e-mail
- Triggering an action to the Data Synchronization Service if the user has new e-mail

The **inbox cache** is a local image of a user inbox. This image is used to filter e-mails to be sent to the client and contains the message ID and a few e-mail headers. It does not contain sensitive information or the body of the e-mails. For more information on inbox cache, see Chapter 6.

For more information on the life cycle of a message, please see section *Funambol push*.

PIM Connector

This connector is the counterpart of the E-mail Connector for **Personal Information Management** (**PIM**) data synchronization.

Personal Information Management data are, for example, the address book, appointments, events, tasks, and so on.

The PIM Connector is the component that allows the Data Synchronization Service to interact with the PIM database to synchronize devices connected to the Funambol Server. Out of the box, Funambol provides its own PIM database. However, as we will see in the following chapters, Funambol can be extended to integrate any external PIM datasource.

The PIM Connector is deployed with the synchronization engine and has the following responsibilities:

- Searching the PIM items that the user has modified on the server since the last synchronization
- Keeping the client updated with the server's data

For more information on the lifecycle of a PIM update, please see the section *Funambol push*.

PIM Listener Service

The PIM Listener Service monitors the user PIM database and detects if a user has unsynchronized changes. It is a separate process from the Data Synchronization Service and has the following responsibilities:

- Detecting updates in the PIM database to be pushed to the user device
- Triggering an action to the Data Synchronization Service if the user has PIM changes

For more information on the life cycle of a message, please see the section *Funambol push*.

Push Connection Service

The Push Connection Service is a separate process from the Data Synchronization Service and is responsible for the implementation of the server-side, connection-oriented push technology, which is described in the section *Funambol push*.

The main responsibilities of the Push Connection Service are:

- Accepting and keeping open connection-oriented push connections from devices
- Delivering push notifications to the attached devices

Web Load Balancer

When deploying systems for high availability and reliability, it is common to have redundant components that handle requests from clients. This allows better scaling of the system as the number of users increases and to better handle failure conditions should a system crash or stop working.

SyncML is an application protocol transported over HTTP. This means SyncML requests can be considered as common HTTP traffic. A Web Load Balancer can therefore be used to balance the incoming load among different nodes of a Data Synchronization Service cluster. In this way, all nodes in the cluster perform the same function and can be used interchangeably for each SyncML request.

Note that in the architecture illustration only one component of each service is shown (that is, one DS Service, one Push Connection Service, one PIM Listener Service, one Inbox Listener Service), but each component can actually be redounded in a so called **cluster**.

The main responsibilities of the HTTP Load Balancer include:

- Providing the frontend of the Funambol system
- Distributing SyncML requests among the nodes of the Data Synchronization Service cluster
- Detecting node failures within the cluster and redirecting traffic to the remaining active nodes when a node fails

The HTTP Load Balancer is not provided as part of the default installation or deployment. Many different solutions, both hardware and software, can be adopted. Organizations may already have different best practices in place. A common solution is to use Apache (`http://en.wikipedia.org/wiki/Apache_http`) along with mod_jk (`http://en.wikipedia.org/wiki/Mod_jk`).

IP load balancer

Most Funambol Sync Clients for mobile devices use a TCP or IP-based push mechanism that requires a TCP/IP connection from the client to the server to be constantly open (in particular, the Funambol Push Connection Service). This means the number of connections that the connection service must support is directly proportional to the number of users using push. This type of push is called **connection-oriented push**, as described in the following *Funambol push* section.

When the number of these connections becomes higher than a single machine can sustain, an IP load balancer is needed. This component works similar to the Web Load Balancer, but at the Transport level (Layer 4 load balancing, see `http://kb.linuxvirtualserver.org/wiki/Load_balancing`).

The main responsibilities of the IP load balancer are:

- Providing the frontend of the Funambol system for connection-oriented push requests
- Distributing connection-oriented push requests among the nodes of the Push Connection service cluster
- Detecting failures on the nodes of the cluster, redirecting traffic to the active nodes if a node fails

The IP Load Balancer is not provided as part of the default installation or deployment. Many different solutions, both hardware and software, can be adopted. Many organizations may already have different best practices in place. A common solution is to use a Linux Virtual Server (`http://en.wikipedia.org/wiki/Linux_Virtual_Server`).

Database

When Maria installed Funambol on her desktop computer, the setup package installed a personal database that the server used to store her personal data. Maria's IT department, though, has a dedicated database server and DBA that takes care of the database needs of departmental applications. This database server is represented in the illustrated architecture graph by the **Database** box.

Funambol is tested with and supports the following open source database systems:

- PostgreSQL
- MySQL

However, potentially it is compatible with any JDBC-enabled database.

E-mail Provider

This component is not part of Funambol software. It represents the users' e-mail servers. Currently, the supported mail protocols are:

- IMAP
- POP
- Google IMAP
- AOL IMAP

Funambol push

In Chapter 1, Maria manually synchronized Mozilla Thunderbird and her mobile phone using the Funambol Mozilla Sync Client and the native SyncML client on her phone. With Funambol, there is an easier way to keep Maria's personal information in sync—push synchronization. Using this method of synchronization, Maria does not need to manually start the synchronization to receive the changes made with another application or device. The push can be bidirectional:

- **Client-to-server**: In this case, a Funambol Sync Client detects when entries of the address book or calendar on the device have changed and, if necessary, automatically synchronizes the updates to the server
- **Server-to-client**: In this case, the client receives a notification from the server that there is new data to synchronize and initiates a new synchronization

Note that in the case of server-to-client synchronization, the server just notifies the client that a new synchronization should take place, no data exchange is performed. The synchronization is always started by the client.

Another synchronization method that does not require user interaction is the **scheduled sync**. This type of synchronization may be available on a wider range of devices in addition to the ones with a Funambol Sync Client, but it is not a real push. In this case, the client synchronizes on a regular basis (for example, every hour) with the server regardless of whether there are updates to exchange or not.

From a system perspective (which is what Maria is interested in), to deploy Funambol in her company, Maria needs server-to-client push, which requires the systems described earlier to be set up. The next section provides a better insight of this push technology.

Server-to-client push

Funambol server-to-client push is based on the delivery of a **push notification**, which is a small packet of data that Funambol sends to a device for triggering a new synchronization. The notification package is technically called PKG#0 and contains information such as:

- The server that is requesting a synchronization and the datasource to be synchronized.
- The type of synchronization to be performed. PKG#0 can be delivered in many different ways. One of the most common ways is using a TCP/IP connection, which is possible in two flavors—connection-less and connection-oriented.

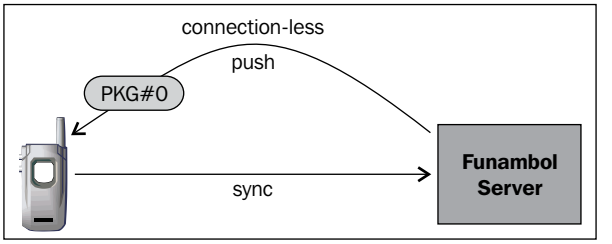
Each of these present advantages and disadvantages, and due to their intrinsic modes of operation, not all techniques can be applied on all TCP/IP networks.

Further, not all devices and clients implement the same push functionality. For technical reasons, many do not support it or use different technologies. For example, Maria's Nokia phone does not support TCP/IP push natively (you need a Funambol extension—the Funambol Symbian Sync Client—to do it).

Out of the box, Funambol detects the most appropriate push mechanism for the given client or device based on clients and network capabilities, and uses it.

Connection-less push

As illustrated in the following figure, connection-less push is based on the availability of the device with a routable IP address, that is, an IP address that the server can reach by opening a socket connection from the server to the client. With this mechanism, PKG#0 is delivered by the server through a TCP/IP connection to the device that reads the content of the push notification. The device then starts a new synchronization for the datasources specified in PKG#0.



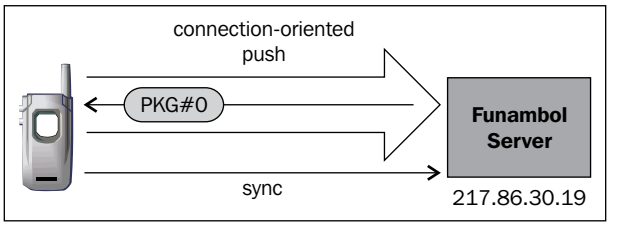
For connection-less push to work, the server needs to know the IP address of the mobile device, which very likely changes over time. For example, in a GPRS/UMTS network, a mobile phone gets a new IP address at each connection to the network. This happens, for example, when the device is turned on or if the connection is re-established after the phone was out of coverage.

Connection-less push requires the ability by the Funambol Sync Client to detect the IP address of the device every time it changes so that the client can communicate it to the server. In other words, when the Funambol Sync Client detects that a device (such as a mobile phone) has a new IP address, it starts a connection to the server to communicate the new address. From that moment onwards, the server will use the new address to contact the device and deliver the push notification. The device address is stored in a portion of the Funambol database called **Device Inventory**, so as to be used when a new PKG#0 shall be delivered to the device.

Not all carriers provide full Internet access and therefore do not provide a public and routable IP address. In particular, this mechanism usually does not work with WAP connectivity, as this type of connection requires all traffic to be routed through a WAP gateway. For this reason, the network devices attached to it are in a private network that is not directly reachable from the Internet. In these cases, connection-oriented push will be used.

Connection-oriented push

Connection-oriented push works the opposite way to connection-less push. As illustrated in following figure, the device starts a client-to-server connection and keeps it alive, waiting for a new PKG#0 to be read. Once the server writes a new push notification into such a connection, the client wakes up and reads it, starting the synchronization as in the case of connection-less push. If, for some reason, the connection drops, the device tries to re-establish it as soon as possible.



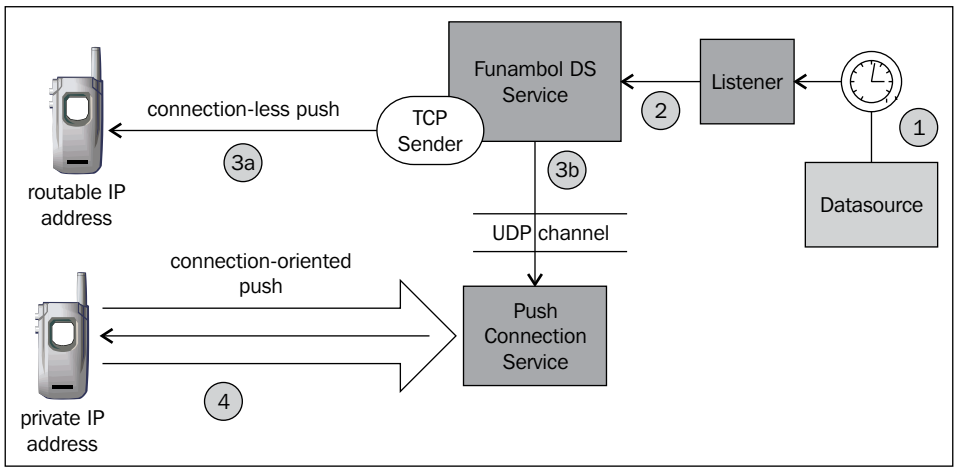
Connection-oriented push does not require the device to acquire a routable IP address from the carrier network. If the device is able to connect to the Internet, it should not have any issue in using this push method.

A disadvantage of connection-oriented push is that it consumes more resources than connection-less push, as the connection must stay open all the time. Additionally, to overcome timeout issues with specific networks and gateways, a heartbeat packet is sent to the server on a regular basis. This is a small 4 byte packet that tells the network that the connection is still alive, and which is sent every 5 minutes. Even if mobile networking is optimized to use as few resources as possible, this usually results in a an increased drain on mobile phone batteries.

Life cycle of a push notification

So far, Maria has gathered a lot of information about how Funambol works from a system perspective. Now she knows which kind of systems she needs to plan and set up and how they interact. She also understands the networking aspects behind an enterprise Funambol deployment.

Still, Maria may not be completely clear on the flow of push synchronization (server-to-client push), from the time a change is detected to when the change is synchronized on the mobile device. This is illustrated in the following figure:



As described earlier, the role of the Listener services is to detect changes in a datasource (for example, a mail server, the Funambol PIM database, an enterprise CRM system, and so on). For the sake of simplicity, let's assume that the datasource box in the previous figure represents Maria's e-mail service and that the Listener box represents the Inbox Listener. In its standard configuration, the Listener performs the following two tasks:

- The listener polls user inboxes on a regular basis to check if there is any new e-mail
- If new e-mail is detected, the Listener tells the Data Synchronization Service to send a PKG#0 to the specific user or to a particular device

Once the server is aware that a particular user, for example Maria, must be notified that there is something to synchronize, it determines which clients are associated with the user and which push technology to use based on the device type. For instance, if it is a Java ME capable device or a BlackBerry device running the Funambol E-mail Client, connection-oriented push will be used; if the device is a Windows Mobile or Symbian device, connection-less push will be used by default; if the IP address of the device is not routable, then connection-oriented TCP push will be used.

The notification message is delivered through a component called *Sender*, which is part of the Data Synchronization Service.

- First of all, the DS Service retrieves the device's IP address from the device inventory and tries to deliver the package using a connection-less push call through a TCP Sender.
- If the IP cannot be reached, the server tries connection-oriented push. In this case, the server sends a UDP packet to tell the Push Connection Service that the device should be notified.
- The Push Connection Service checks if the device to be notified is currently connected to the Push Connection Service. If so, the PKG#0 is delivered. If not, the UDP packet is discarded.

For connection-oriented push to work, UDP traffic between the Data Synchronization Service and the Push Connection Service must be enabled. As there can be more Push Connection Services running on a cluster of servers, this traffic is based on IP multicast (see `http://en.wikipedia.org/wiki/IP_multicast`), which must be enabled at the networking level.

Likely, Maria owns a Symbian phone on which connection-less push works. This is great because this will save her phone's battery.

After the PKG#0 has been received and processed, the device will start the synchronization of the datasources that have changed, according to the content of the notification package without letting Maria even notice it.

Summary

In this chapter, Maria acquired detailed information on the Funambol architecture, which systems it is based on, and how Funambol services interact with each other. She also learned about the network requirements for deploying Funambol in her company along with the different push mechanisms and how they work. This is the basis to synchronize all of the different devices that Maria's colleagues may have, with a central PIM database and e-mail system.

Maria is now ready to master the configuration of each component and service, which will be covered in the next chapters.

3
Mastering Data Synchronization

In the previous chapters, Maria learned how flexible a Funambol deployment can be, ranging from personal use to an enterprise deployment that can scale up to thousands of users. Maria learned about the different Funambol components, the requirements of each component in terms of system resources, and the role that each component plays in a complex deployment.

This chapter covers the most important configuration changes that Maria may apply in her company deployment. To understand the concepts behind certain setup options, the following sections provide details on how Funambol handles configuration settings. In fact, the deployment flexibility of Funambol requires each component to be flexibly configured so that, with simple changes to a few parameters, it is possible to support different deployment scenarios. This is achieved with a flexible configuration mechanism described in this chapter.

Even if a component provides many configuration parameters, not all of the parameters need to be exposed to a system administrator, at least in common situations. Funambol engineers have put a lot of work into simplifying things as much as possible to be able to configure the software without coding.

As real-life deployments are likely to be performed on Linux systems, in the following sections, we assume that Maria has installed Funambol on one of the most common Linux distributions. Windows users can use the descriptions in the following sections as a reference and perform similar tasks using the Windows command line.

Starting and stopping the Data Synchronization Server

In Chapter 1, Maria started all Funambol services at once with the following command:

```
$FUNAMBOL_HOME/bin/funambol start
```

If different services need to be started on different machines, Maria can use the starting scripts provided for each Funambol service.

The Data Synchronization Service can be started with:

```
$FUNAMBOL_HOME/bin/funambol-server start
```

Launching the command without parameters shows all the available options:

```
$ bin/funambol-server
Usage: funambol-server command
command:
  start           Start the server
  stop            Stop the server
  license         Show the license
```

To stop the server, Maria can execute the command with the stop parameter.

```
$FUNAMBOL_HOME/bin/funambol-server stop
```

The parameter `license` displays the terms under which Funambol is licensed (AGPL 3.0 for Funambol v7.1).

Funambol services are based on the Java™ platform, therefore some aspects of the **Java Virtual Machine** (**JVM**) can be influenced by parameters given on the command line when the JVM is invoked.

There are three important environment variables that Maria can use to control the behavior of JVM—`JAVA_HOME`, `JAVA_OPTS`, and `MEM_OPTS`.

`JAVA_HOME` can be used to point to a specific JVM. Funambol installs a version of the Java Runtime Environment that works out of the box. Maria can use a different Java implementation just by changing the `JAVA_HOME` variable.

Standard options of a JVM can be set in the `JAVA_OPTS` environment variable before starting Funambol.

One parameter that generally changes from one machine to another is the amount of memory available for running programs. The JVM, by default, does not use all of the available memory, instead it must be instructed with the maximum amount of RAM to use. To do this, Maria can set the `MEM_OPTS` environment variable.

For example, if Maria wants to install the Data Synchronization Service on a Red Hat Linux machine with 4 GB of RAM and the latest version of the JVM installed, she can use the following commands:

```
$ export JAVA_HOME=/opt/jdk-1.6.0.13
$ export MEM_OPTS="-Xmx3G"
$ bin/funambol-server start
```

The first command sets the JVM to use. If no `JAVA_HOME` is defined, Funambol defaults to the one distributed with the package, which is `$FUNAMBOL_HOME/tool/jre-1.5.0`.

The second command sets the maximum Java heap size (`-Xmx`) to 3 GB.

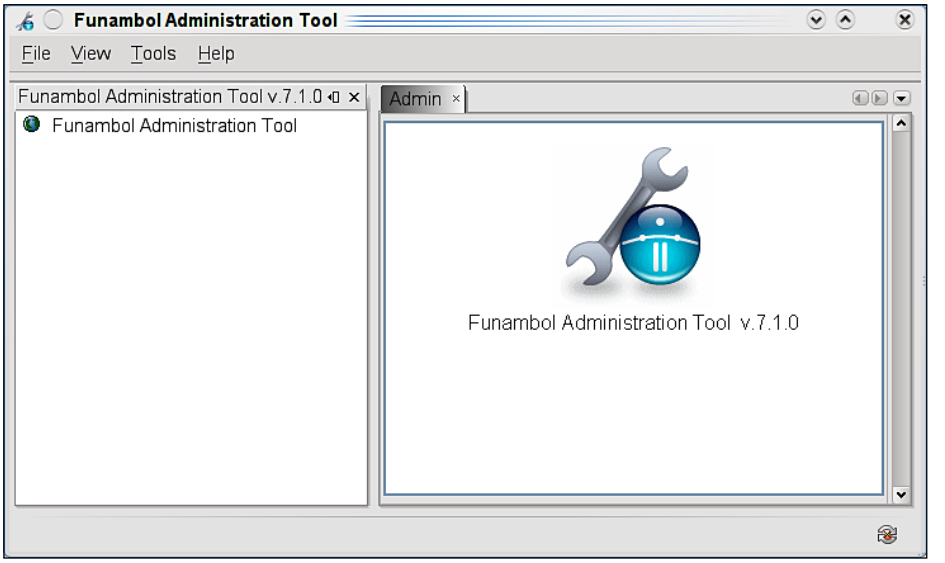
For more information about the `JAVA_HOME` environment variable and the JVM options that can be set with `JAVA_OPTS` and `MEM_OPTS`, please see the Java Virtual Machine documentation (see `http://java.sun.com/javase/technologies/hotspot/vmoptions.jsp`).

General server settings using the Funambol Administration Tool

An application that Maria can use to make many configuration changes in an easy way is the Funambol Administration Tool. This is a client application that runs remotely on a desktop, connects to a Funambol server, and allows the inspection and change of many configuration settings. The Funambol Administration Tool can also be run locally on the server where Funambol is installed if a graphical interface (that is X11) is installed, as the standard distribution already contains the tool. To launch it on Linux, Maria can simply open a terminal, go into the `$FUNAMBOL_HOME` directory, and type:

```
$ admin/bin/funamboladmin
```

The following screenshot will appear:

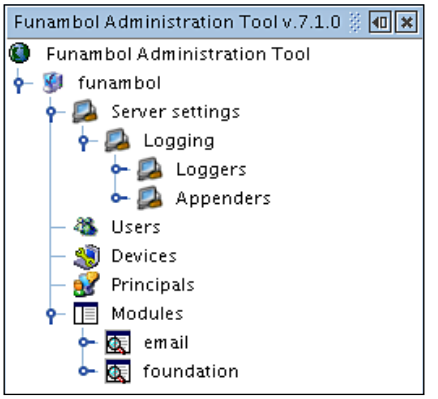


To start the Funambol Administration Tool on Windows, select **Funambol Administration Tool** from the **Start** | **All Programs** menu.

Before being able to see or change anything, Maria needs to log in to the Funambol server. All she needs to do is select **File** | **Login** from the Administration Tool menu and enter her credentials in the login screen. The default connection information for a fresh Funambol installation server is:

```
Hostname/IP: <hostname>
Port: 8080
Username: admin
Password: sa
```

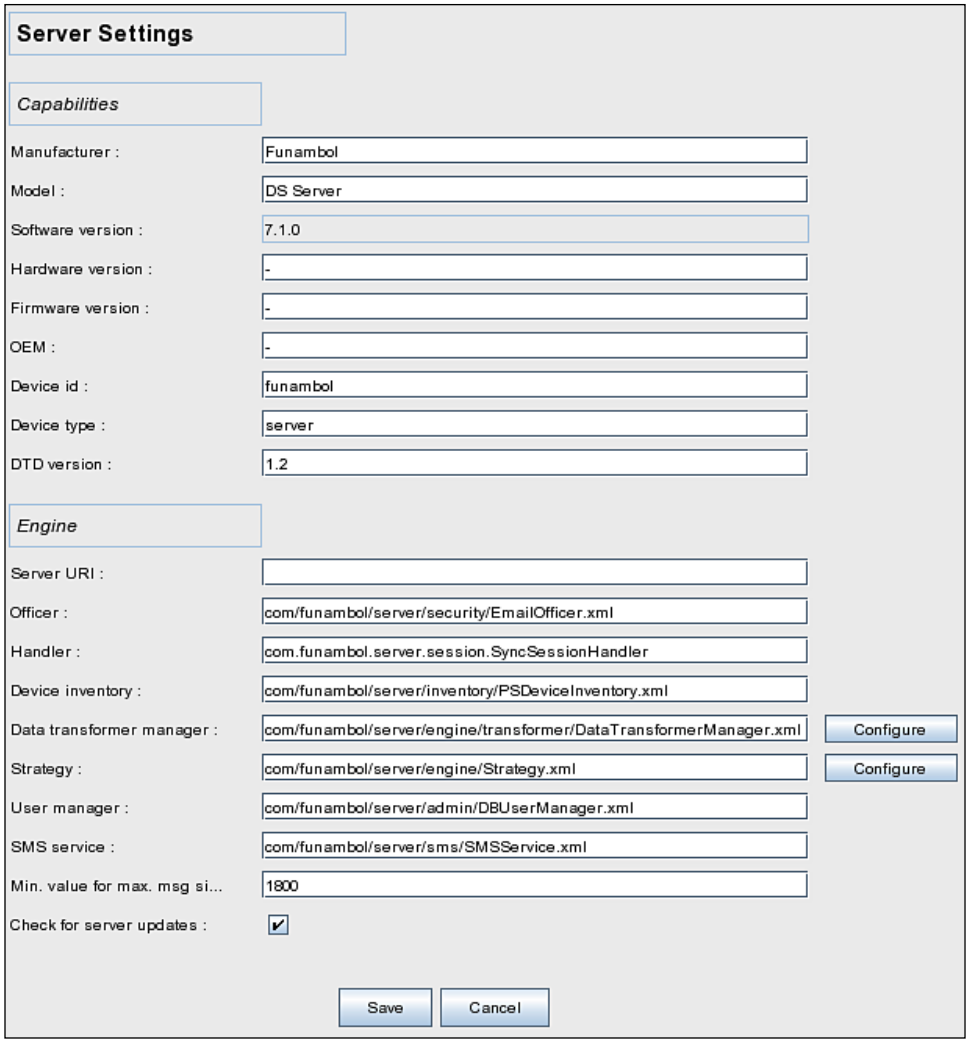
After logging in and expanding the tree in the left pane of the Administration Tool, a structure as shown in the following screenshot is displayed:



From the configuration tree, Maria can access many aspects of the Funambol server. Each of these is described in more detail in the following sections.

Server settings

Double-clicking the node **Server settings**, the Funambol Administration Tool displays the following form in the right pane of the main window:



A description of the properties that can be changed in this form is provided in the next sections.

Server capabilities

The first group of settings is used to identify the server, the version of the software it is running, and the service provided. Based on the fact that in the SyncML domain, the server is a specific kind of device, the information provided by the Server Settings is called device capabilities. All properties in the Server Settings section are used to provide SyncML clients with the server capabilities as defined by the SyncML protocol.

Note it is very unlikely that Maria will need to change these values unless she is embedding the Funambol server in another service. However, it is worthwhile to provide a short description of each parameter as they may be needed for troubleshooting user or configuration issues later.

Property	Description
Manufacturer	The vendor of the server
Model	A model tag representing the kind of service provided (for example, OMA DS server)
Software version	The version of the server
Hardware version	Not used
Firmware version	Not used
OEM	Field available to OEM (for example, Acme CRM)
Device ID	The unique identifier of the server
Device type	It specifies that this is a server
DTD version	The minimal SyncML DTD version supported by the server

To change one of the above parameters, just edit the value and click **Save**.

Engine settings

The second group of settings, Engine settings, contains more interesting information. Again, in the case of a default deployment, Maria does not need to be concerned about these values. These settings, however, may need to be changed if the server is extended with new connectors. For now, Maria does not plan to develop any extensions to the server; she just needs to know what these settings are for, without going into further detail.

Available engine settings are described in the following table:

Property	Description
Officer	Represents the component that controls how users are authenticated and granted access to the system.
ServerURI	Specifies the URI to be used by the client in responding to server messages. In SyncML, each response from the server may contain, in a parameter called `respURI`, the URL to which the client must send the next request. This is used to attach session information to the URL the client will use in the next call so that a load balancer can use sticky sessions to direct the call to the proper node of a cluster of servers.
SessionHandler	Represents the component that manages the synchronization session.
Strategy	Represents the component that handles the synchronization process.
UserManager	Represents the component that handles access to the user storage.
Min. value for max. msg size	Specifies the minimum value of `MaxMsgSize` the server supports. MaxMsgSize is a value given by the clients to tell the server not to send messages bigger than a certain size. If this value is too small, the server may be unable to create a response at all. Therefore, if a client specifies a `MaxMsgSize` smaller then **Min. value for max msg size**, the server refuses the synchronization.

Property	Description
Device inventory	Represents the component that manages an inventory of user devices.
Datatransformer manager	Represents the component that allows to customize the data transformations applied to incoming and outgoing messages.
Check for server updates	Enables the Data Synchronization Service to check the Funambol website if there is a new version of Funambol available.

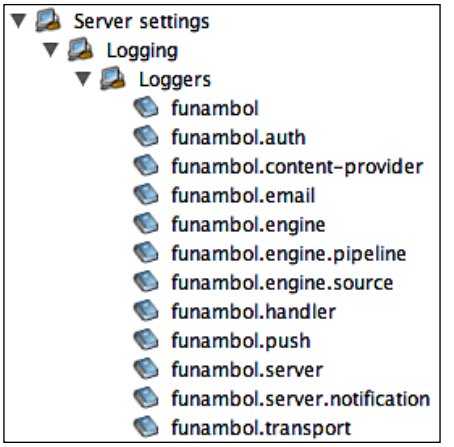
To change one of the above parameters, just edit the value and click **Save**.

Logging

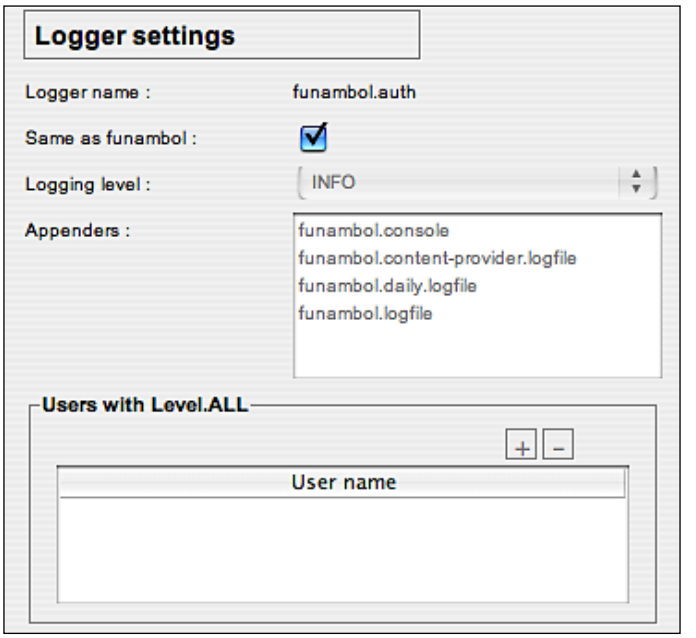
As Maria will operate the Funambol installation in her company, one of her main concerns is how she can monitor what is going on, if there are errors and so forth. The best way to address these questions is through Funambol's logging capabilities. As Maria has seen in earlier chapters, a Funambol deployment consists of several components. Further, Funambol can be extended with additional connectors, each needing to log information.

To avoid mixing everything into a single flow of information, Funambol categorizes log information using a namespace that associates the information that is logged with the area of the product that generated it. These namespaces are called **loggers**.

Maria can access the loggers available in the server by expanding the **Server settings | Logging** node in the **Funambol Administration Tool**. She will get a list similar to the one depicted in the following illustration:



Logger names are organized hierarchically so that an inner logger inherits its settings from the parent logger. For example, the logger `funambol.auth` inherits its configuration from the top-level logger `funambol`. Double-clicking a logger displays a panel as shown in the following screenshot:



Each logger has a name, a logging level, a list of appenders, and a list of users that must be logged at the **ALL** logging level, regardless of the specified level for the logger.

Each logger is specialized in a specific role, as described in the following table:

Logger	Description
`funambol`	The parent of all funambol loggers. Changing the configuration of this logger applies to all child loggers.
`funambol.auth`	Logger for authentication events.
`funambol.content-provider`	Logger for events generated by the content provider.
`funambol.email`	Logger for events generated by the e-mail connector.
`funambol.engine`	Logger for events generated by the engine.
`funambol.engine.pipeline`	Logger for events generated by the input and output pipelines.
`funambol.engine.source`	Logger for events generated by the sync sources.
`funambol.handler`	Logger for events generated by the synchronization session handler.
`funambol.push`	Logger for events generated by the push service.
`funambol.server`	Logger for events generated by the DS service.
`funambol.server.notification`	Logger for events generated by the notification engine of the DS service.
`funambol.transport`	Logger for events generated at the transport layer.

Additional connectors may install more loggers. In this case, they will be listed in the **Funambol Administration Tool** panel together with the standard Funambol loggers.

Logger settings

Each logger can be configured separately or can inherit its configuration settings from a parent logger. By default, all child loggers of the `Funambol` logger inherit from it.

If Maria wants to override the parent settings for a logger, all she needs to do is uncheck **Same as <parent logger>** and apply the desired settings. The meaning of each setting is described as follows:

- **Logger name**: This is the logger name as described earlier.
- **Logging level**: This is the amount of detail provided by a logger. The lower the level, the more detailed the information that the logger will record. The lowest level is **TRACE**, which is used mostly to log information during development. The highest is **FATAL**. All possible values are:
    - **OFF**: No logging
    - **FATAL**: Fatal errors
    - **ERROR**: Recoverable and unrecoverable errors
    - **WARN**: Warning
    - **INFO**: Informative messages for an administrator
    - **DEBUG**: Debug information
    - **TRACE**: Low level development information
    - **ALL**: Log everything

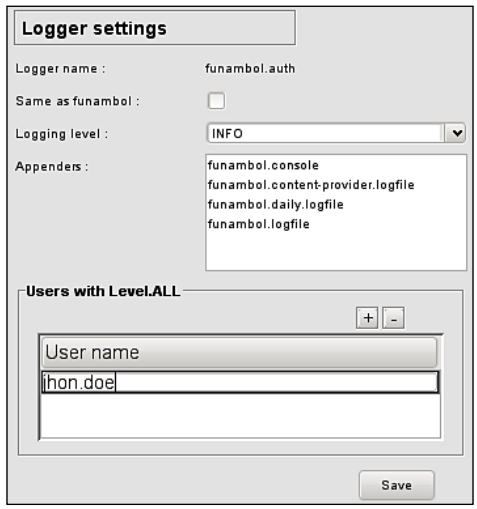
  The default logging level for all loggers is **INFO**.

- **Appenders**: This is a list of log appenders. Log messages are sent to each appender of this list to be output to different media (the console, a file, a database, and so on). A detailed description of appenders is provided in the following sections.
- **Users with Level.ALL**: This is a list of users for which the log level must be **ALL** regardless of the specified level for the logger. This option is very useful to troubleshoot issues specific to a particular user without impacting the global logging settings.

Logging the activity of a single user

Maria, as administrator of the system, will need to troubleshoot user issues or verify conditions for a specific user. To do so, she can set the log level of a user to ALL. This means all user activity will be logged at the normal level for each appender, but the activity for the specified users will be logged in detail.

To enable full logging for a user with **User name** as **john.doe** for the `funambol.auth` logger, Maria simply needs to double-click the logger name to display its settings in the right pane, as shown in the following screenshot:



Then she needs to uncheck **Same as funambol** to override the standard settings and click the small **+** button in the **Users with Level.ALL** box. A new entry appears in the list below the button.

The last step is to click the new entry and type the username of the user to monitor and click **Save**.

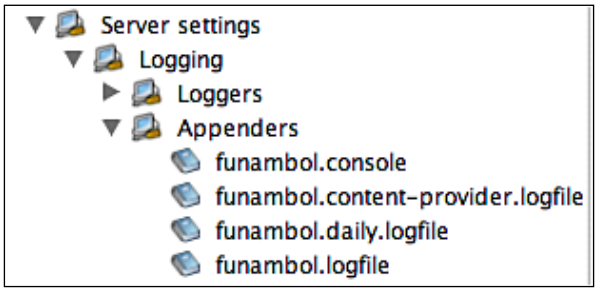
The new setting is picked up by the server automatically. After this change, Maria will note that the authentication activity of that user appears at the highest level of detail in the log files.

Appenders

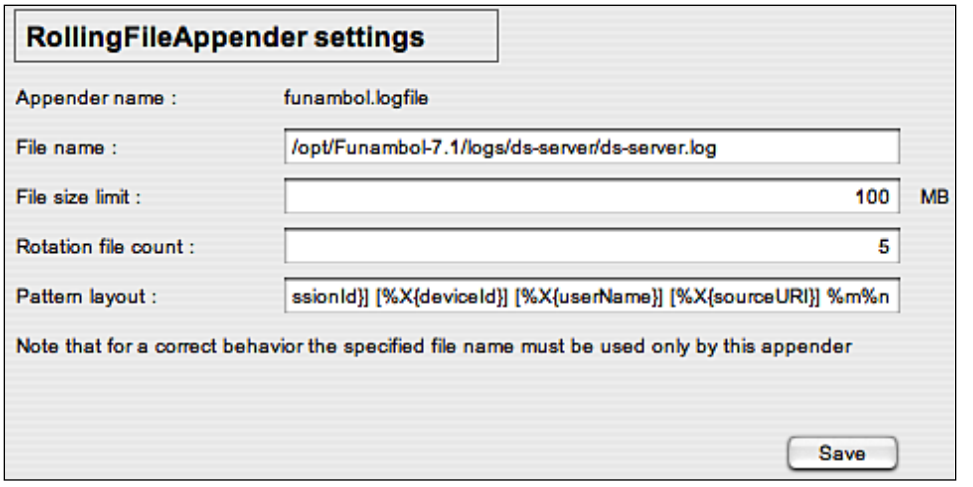
As mentioned earlier, loggers send log events to a chain of appenders. Each appender traces log events in a specific way. Funambol provides the appenders described in the following table, but like loggers, Funambol extensions and connectors may add their own.

Appender	Description
`funambol.console`	Writes events on the console.
`funambol.content-provider.logfile`	Writes events in the log file for the `content-provider` component. This is an appender specifically designed to be used by the content provider so that it can be easily accessed.
`funambol.daily.logfile`	Writes events in a daily rotating file.
`funambol.logfile`	Writes events in a rotating file as per the given properties (file size limit and number of files).

Appenders can be configured individually by expanding the **Appenders** node in the configuration tree as shown in the following screenshot:



The default configuration of the appender `funambol.logfile` is illustrated in the following screenshot:



Note that all appender configuration panels have an entry labeled **Pattern layout** to specify the layout of each log entry. The format of the layout is described in the following section.

Log entry pattern

The log entry pattern is a string of text that specifies the format of a log entry using placeholders for information that will be replaced when the entry is written.

Each placeholder starts with a percent sign (%) and is followed by optional format modifiers and a conversion character. The conversion character specifies the type of data—for example, category, priority, date, and thread name. The format modifiers control things such as field width, padding, and left or right justification.

The following is a simple example. If the conversion pattern is `%-5p [%t]: %m%n`, a possible log entry in the appender would be similar to:

```
DEBUG [main]: Message 1
WARN  [main]: Message 2
```

Note that there is no explicit separator between text and placeholders. When the pattern parser reads a conversion character, it knows when it has reached the end of a placeholder.

In the previous example, the placeholder `%-5p` means the priority of the logging event should be left justified to a width of five characters. The recognized conversion characters are listed in the following table:

Conversion character	Effect
`c`	Used to output the category of the logging event. The category placeholder can be optionally followed by a precision specifier, that is, a decimal constant in brackets. If a precision specifier is given, only the corresponding number of rightmost components of the category name is printed. By default, the category name is printed in full. For example, for the category name `a.b.c` the pattern `%c{2}` will output `b.c`.
`C`	Used to output the fully qualified class name of the caller issuing the logging request. This placeholder can be optionally followed by a precision specifier that is a decimal constant in brackets. If a precision specifier is given, only the corresponding number of rightmost components of the class name will be printed. By default, the class name is output in fully qualified form. For example, for the class name `org.apache.xyz.SomeClass`, the pattern `%C{1}` will output `SomeClass`. **WARNING**: Generating the caller class information is slow. Thus, its use should be avoided unless execution speed is not an issue.

Conversion character	Effect
d	Used to output the date of the logging event. The date placeholder may be followed by a date format specifier enclosed between braces. For example, `%d{HH:mm:ss,SSS}` or `%d{dd MMM yyyy HH:mm:ss,SSS}`. If no date format specifier is given, then ISO8601 format is assumed. The date format specifier follows the same syntax as the time pattern string of the `SimpleDateFormat`. Although part of the standard JDK, the performance of `SimpleDateFormat` is poor. For better results, it is recommended that you use the `log4j` date formatters. These can be specified using one of the strings `ABSOLUTE`, `DATE`, and `ISO8601` for specifying `AbsoluteTimeDateFormat`, `DateTimeDateFormat`, and `ISO8601DateFormat` respectively—for example, `%d{ISO8601}` or `%d{ABSOLUTE}`. These dedicated date formatters perform significantly better than `SimpleDateFormat`.
F	Used to output the file name where the logging request was issued. **WARNING**: Generating caller location information is extremely slow. Its use should be avoided unless execution speed is not an issue.
l	Used to output location information of the caller who generated the logging event. The location information depends on the JVM implementation but usually consists of the fully qualified name of the calling method, followed by the caller's source file name and line number between parentheses. The location information can be very useful. However, its generation is extremely slow. Its use should be avoided unless execution speed is not an issue.

Conversion character	Effect
`L`	Used to output the line number from where the logging request was issued. **WARNING**: Generating caller location information is extremely slow. Its use should be avoided unless execution speed is not an issue.
`m`	Used to output the application supplied message associated with the logging event.
`M`	Used to output the method name where the logging request was issued. **WARNING**: Generating caller location information is extremely slow. Its use should be avoided unless execution speed is not an issue.
`n`	Outputs the platform-dependent line separator character or characters. This conversion character offers practically the same performance as using non-portable line separator strings such as `\n`, or `\r\n`. Thus, it is the preferred way of specifying a line separator.
`p`	Used to output the priority of the logging event.
`r`	Used to output the number of milliseconds elapsed from the construction of the layout until the creation of the logging event.
`t`	Used to output the name of the thread that generated the logging event.
`x`	Used to output the **nested diagnostic context** (**NDC**) associated with the thread that generated the logging event.
`X{sessionId}`	Used to output the session identifier when applicable and available.

Conversion character	Effect
`X{deviceId}`	Used to output the device identifier that generated the log entry when applicable and available.
`X{userName}`	Used to output the user name associated with the session when applicable and available.
`X{sourceURI}`	Used to output the datasource being synchronized when applicable and available.
`%`	The sequence %% outputs a single percent sign.

By default, the relevant log information is printed without formatting. However, with the help of format modifiers, it is possible to change the minimum field width, maximum field width, and the justification.

The optional format modifier is placed between the percent sign and the conversion character.

The first optional format modifier is the **left justification flag**, which is just the minus (-) character. Then comes the optional **minimum field width** modifier, which is a decimal constant that represents the minimum number of characters to output. If the data item requires fewer characters, it is padded on either the left or right until the minimum width is reached. The default is to pad on the left (right justify), but you can specify right padding with the left justification flag. The padding character is space. If the data item is larger than the minimum field width, the field is expanded to accommodate the data. The value is never truncated.

This behavior can be changed using the **maximum field width** modifier, which is designated by a period followed by a decimal constant. If the data item is longer than the maximum field, the extra characters are removed from the *beginning* of the data item and not the end. For example, if the maximum field width is eight and the data item is ten characters long, the first two characters of the data item are dropped.

The following table shows examples of various format modifiers for the category placeholder:

Format modifier	Left justify	Minimum width	Maximum width	Comment
`%20c`	false	20	none	Left pad with spaces if the category name is less than 20 characters long.
`%-20c`	true	20	none	Right pad with spaces if the category name is less than 20 characters long.
`%.30c`	NA	none	30	Truncate from the beginning if the category name is longer than 30 characters.
`%20.30c`	false	20	30	Left pad with spaces if the category name is shorter than 20 characters. However, if category name is longer than 30 characters, truncate from the beginning.
`%-20.30c`	true	20	30	Right pad with spaces if the category name is shorter than 20 characters. However, if category name is longer than 30 characters, truncate from the beginning.

The default layout used by Funambol is as follows:

```
[%d{yyyy-MM-dd HH:mm:ss,SSS}] [%c] [%p] [%X{sessionId}]
[%X{deviceId}] [%X{userName}] [%X{sourceURI}] %m%n
```

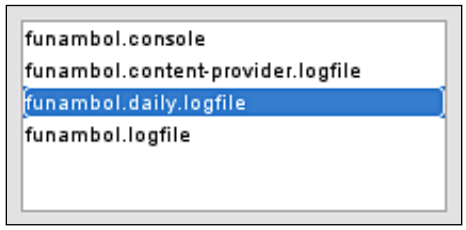
More information about the above patterns can be found at: `http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html`.

Logging configuration examples

We will now show Maria some examples of changes that she can apply to reconfigure the default Funambol logging system.

Using daily rotating log

If Maria wants to have a log that is rotated every day, she can use the `funambol.daily.logfile` appender. The quickest way to do this is to open the **Funambol Administration Tool** and display the settings for the funambol logger. Select the daily logger in the list of the available loggers, as shown in the following screenshot:



Click on **Save** and Funambol will pick up the new configuration after a few seconds.

With this appender, Maria will notice that in `$FUNAMBOL_HOME/logs/ds-server`, a new file named `ds-server.daily.log` has been created. Its content will be similar to:

```
[2009-05-25 10:53:17,766] [funambol.configuration]
[INFO][83B5AC4BDAF8A2A83751642CCDAFDC49] [] [] []
Logger 'funambol' saved successfully
[2009-05-25 10:53:17,782] [funambol.transport.http]
[INFO] [83B5AC4BDAF8A2A83751642CCDAFDC49] [] [] []
Closing session 83B5AC4BDAF8A2A83751642CCDAFDC49
```

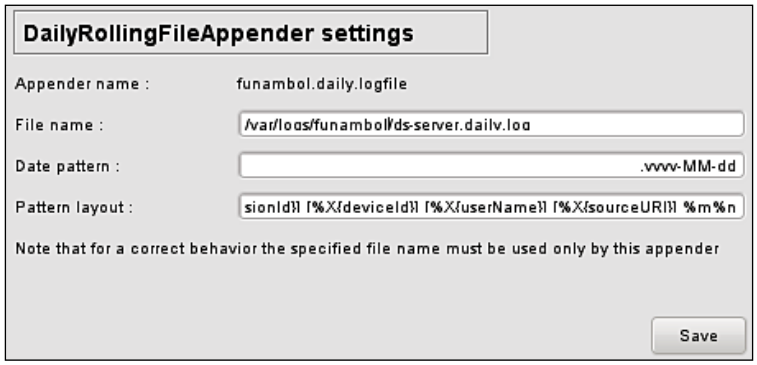
This file will be rotated on a daily basis with files of the form `ds-server.daily.log.YYYY-MM-DD` (for example, `ds-server.daily.log.2009-05-25`).

Changing the location of log files

Changing the location of log files is straightforward in majority of the cases. All Maria needs to do is change the settings of the appender used by the loggers. If all loggers inherit from Funambol, it is sufficient to update the appender used by the Funambol logger. By default, this is the `funambol.logfile` appender, unless Maria changed it as explained earlier.

To change the settings of the appender, double-click it and change the file name pattern.

Consider that Maria wants to store daily logs in `/var/logs/funambol`. After creating the directory and making sure it has the proper permissions for the Funambol process, she can display the appender configuration panel and save the new settings as shown in the following screenshot:



This change affects all loggers that inherit from the main logger.

Changing the log entry pattern

Logs are a good tool for monitoring system health. In many organizations, this is done with background jobs that search log files for text patterns that represent a particular condition such as an error or the start or end of a session, and so on. Another use of log files is to parse them to grab statistical information (such as how many users accessed the system in the last week, how many devices synced, how many synchronizations ended successfully, and so on.).

For example, Maria may desire a more compact log record, with dates in ISO8601 format, no priority, and limiting the logger name just to the innermost portion. To do so, she opens the `funambol.daily.logfile` appender and sets the pattern layout to:

```
[%d{ISO8601}] [%c{1}] [%X{sessionId}] [%X{deviceId}]
[%X{userName}] [%X{sourceURI}] %m%n
```

The new content will be as follows:

```
[2009-05-25 17:06:51,346] [configuration]
[933DDED44D8F57E4717955AC4B27290A] [][] [] Appender
'funambol.daily.logfile' saved successfully
[2009-05-25 17:06:51,352] [http] [933DDED44D8F57E4717955
AC4B27290A]
[] [] [] Closing session 933DDED44D8F57E4717955AC4B27290A
```

Using a remote database

In Chapter 1, the first synchronization that Maria performed did not require a database installation. This is because Funambol Community Edition ships with an embedded database suitable for development or personal use. In the case of a real-life enterprise deployment, the IT infrastructure often provides standardized database services to the other functions in the organization. For example, it is a common practice for database systems to be protected behind a separate network so that they must be accessed remotely. Maria's environment is not an exception with regards to database management.

Funambol database access is based on Java JDBC technology, that is, it should work with any JDBC-compliant database. Funambol directly supports two open source database management systems: MySQL and PostgreSQL.

This section describes how Maria can use either of these databases. In addition to the simple steps necessary to enable Funambol to access a remote database, a more comprehensive description of the Funambol configuration architecture is provided in the following chapters. This will help Maria to better understand how things work and how she can look to customize the Funambol configuration more thoroughly.

Working with the database

It is beyond the scope of this book to cover how a database management system is installed and configured. From here onwards, we will assume Maria has enough database administration experience to configure the database service to meet the requirements described in the following instructions.

Creating and initializing a PostgreSQL database

Following the instructions below, Maria creates a new PostgreSQL database to host Funambol data.

```
createdb -E UNICODE funambol -U postgres
createuser -W
Enter name of role to add: syncuser
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n)
n
Shall the new role be allowed to create more new roles?
(y/n) n
Password: <choose a password>
```

After creating the database, Maria creates the Funambol database schema and imports the initial database data:

```
psql -h <database hostname> funambol -U postgres -f
$FUNAMBOL_HOME/ds-server/default/sql/postgresql/
init_engine.sql
```

Note that with the previous command, Maria created a new database called `funambol` and a new database user called `syncuser` with a password of her choice. This information will be used shortly.

PostgreSQL version 8.3 no longer automatically casts all data types into text strings for functions that take a string input. An explicit cast is needed. This can be accomplished by defining and using a casting function at the end of the initialization script as follows:

```
CREATE FUNCTION pg_catalog.text(bigint)
RETURNS text STRICT IMMUTABLE LANGUAGE
SQL AS 'SELECT textin(int8out($1));';
CREATE CAST (bigint AS text) WITH FUNCTION
pg_catalog.text(bigint) AS IMPLICIT;
```

Creating and initializing a MySQL database

Following the instructions mentioned next, Maria creates a new MySQL database to host Funambol data:

```
create database funambol character set 'UTF8';
create user syncuser identified by 'test';
GRANT ALL PRIVILEGES ON funambol.* TO
'syncuser'@'localhost' IDENTIFIED BY '<choose a
password>';
FLUSH PRIVILEGES;
\q
```

After creating the database, Maria creates the Funambol database schema and imports the initial database data.

```
mysql -h <database hostname> -D funambol –u root <
$FUNAMBOL_HOME/ds
server/default/sql/mysql/init_engine.sql
```

Note that with the previous command, Maria created a new database called `funambol` and a new database user called `syncuser` with a password of her choice. This information will be used shortly.

Configuring Funambol to use the remote database

Different from the other configuration changes that Maria has made, changing the database configuration cannot be done using the Funambol Administration Tool. Instead, Maria needs to change a configuration file stored in the filesystem, which is used by all Funambol components. This file is located in the `$FUNAMBOL_HOME/config/com/funambol/server/db` directory. It is called `db.xml` and is an XML file that Maria can change with any text editor.

Based on the information used earlier to create the database, Maria should make this file look like the following (for PostgreSQL):

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0" class="java.beans.XMLDecoder">
 <object class="com.funambol.server.db.
                DataSourceConfiguration">
   <void method="setProperty">
    <string>url</string>
    <string>jdbc:postgresql://<server database>/
                            funambol</string>
   </void>
   <void method="setProperty">
    <string>driverClassName</string>
    <string>org.postgresql.Driver</string>
   </void>
   <void method="setProperty">
    <string>username</string>
    <string>syncuser</string>
   </void>
   <void method="setProperty">
    <string>password</string>
    <string><password chosen></string>
   </void>
 </object>
</java>
```

Maria will need to replace `<server database>` and `<password chosen>` with the values used during database creation.

For MySQL, the file should look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0" class="java.beans.XMLDecoder">
 <object class="com.funambol.server.db.
               DataSourceConfiguration">
   <void method="setProperty">
    <string>url</string>
    <string>jdbc:mysql:hsql://<server database>/
                              funambol</string>
   </void>
   <void method="setProperty">
    <string>driverClassName</string>
    <string>com.mysql.jdbc.Driver</string>
   </void>
   <void method="setProperty">
    <string>username</string>
    <string>syncuser</string>
   </void>
   <void method="setProperty">
    <string>password</string>
    <string><password chosen></string>
   </void>
 </object>
</java>
```

There is another easy task to do when changing the database server in use. As mentioned, database access is performed through JDBC, and therefore, the proper JDBC driver for the database must be available to the Funambol services. The quickest way to do this is to store the jar of the driver under `JAVA_HOME/jre/lib` of the JVM used to start Funambol. Funambol will pick up the changes at its next startup.

Changing the administrator user's password

One of the configuration changes that Maria needs to make for a real deployment is changing the administrator password. This is the password of the user admin, used earlier in the administration tool.

This can be done by running the `admin-passwd` command and providing the old and new password. For example, from the `$FUNAMBOL_HOME` directory, Maria can type the following command:

```
$ bin/admin-passwd
Changing password for admin
Current password: <sa>
New password: <new password>
Retype new password: <new password again>
```

There is an important implication in changing the administrator password. The same credentials are used by other Funambol services to access the Data Synchronization Service remotely. For example, when a user has a new e-mail, the Inbox Listener calls a remote web service on the DS service to notify it to the user.

For this reason, after changing the administrator password, Maria needs to change a configuration parameter in those services. The affected services are:

- Inbox Listener configuration file: `$FUNAMBOL_HOME/config/com/funambol/email/inboxlistener/InboxListenerConfiguration.xml`
- PIM Listener configuration file: `$FUNAMBOL_HOME/config/com/funambol/ctp/server/CTPServerConfiguration.xml`
- Push connection configuration files: `$FUNAMBOL_HOME/config/com/funambol/pimlistener/PIMListenerConfiguration.xml`

In all three files, the old password needs to be replaced with the new one, as shown in the following code snippet:

```
<object class="com.funambol.server.admin.ws.client.
               ServerInformation">
    <void property="url">
      <string>http://localhost:8080/funambol/services/
                    admin</string>
    </void>
    <void property="username">
      <string>admin</string>
    </void>
    <void property="password">
     <string>new password</string>
    </void>
</object>
```

Using an Apache frontend

A personal installation of Funambol includes all it needs to serve the requests from a SyncML client transported over the HTTP protocol, which means it includes a server capable to serve HTTP requests. This is a configuration suitable for a relatively small environment with limited performance and scalability requirements.

What Maria wants to do instead is install the server inside her organization to serve many thousands of users. A common practice that Maria follows for other web-based services is to have an Apache web server frontend directly interacting with the external world and keeping the business services behind the firewall. Funambol is no different from other services that support this configuration, as described in Chapter 2. The advantages of separating the transport layer service from the other services are:

- It is a best practice
- Apache is a well-known web server; it is a reliable, scalable, and powerful platform that many IT departments are familiar with (which means, Maria does not need to train people in a different web server platform)
- The Apache web server can be easily configured to redirect and rewrite URLs, proxy, and cache content, and to integrate with different backends
- The Apache web server can be easily configured to balance the load among a cluster of backend service providers using the `mod_jk` module

There is a lot of information on how to set up these types of configurations and a detailed explanation of how to configure Apache is beyond the scope of this book. For a step-by-step procedure, Maria can refer to the following wiki page on the Funambol forge: `https://core.forge.funambol.org/wiki/UsingFunambolWithApache`.

Funambol over HTTPS

Another aspect that Maria is concerned about is security, and how to protect the data that is transferred over the network from undesired uses. Another advantage of using the Apache frontend is that once she has configured Apache to serve HTTPS requests (and maybe stopped the HTTP traffic from the outside) and redirect them to Funambol, there is no need to do anything else. Thus, all Maria needs to do is acquire a valid certificate from a trusted certification authority for the hostname she plans to use in production and configure Apache to use HTTPS (for examples visit `http://www.securityfocus.com/infocus/1818`).

The use of a certificate issued by a trusted certification authority (as opposed to use of a self-signed certificate or a certificate issued by an internal certification authority) is not strictly necessary. However, it is recommended as some devices may not recognize as valid certificates that are not trusted.

Funambol configuration concepts

In the previous sections, Maria learned how to make many configuration changes with simple steps. As Funambol is not only a product, but also a platform on top of which third parties or the community can develop more powerful sync applications. It may be helpful to dig into more detail in the Funambol configuration architecture, this may also help Maria to automate some of the most common administration tasks that her IT department has established.

One of the original Funambol design goals was to provide a foundation framework that can be used to implement any kind of synchronization service, extending existing modules, or plugging in new modules. This requires a configuration mechanism that can be easily extended to cover different needs and requirements. Another important requirement that Funambol developers took into great consideration was that such configuration information must be easy to access and modify without the need of special tools. In fact, in many data centers, the only way to access the systems in the production or staging networks is through a remote SSH session.

Many Funambol components are configured by means of a so called server **JavaBean**. The idea is to view a configuration file as the text version of a configuration object that can be used by a service or component like a normal class. This way, any Funambol or third-party component can use the same configuration mechanism. This provides Maria with a common framework to manage all of the configuration parameters, without the need to learn a different configuration tool for each Funambol extension. The other advantage of this approach is that all configuration information is stored in text files that can be accessed and modified with a simple text editor.

This is an example of a server JavaBean:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.4.1_01" class="java.beans.XMLDecoder">
 <object class="sync4j.framework.server.store.
               PersistentStoreManager">
  <void property="jndiDataSourceName">
    <string>java:/jdbc/sync4j</string>
  </void>
  <void property="stores">
   <array class="java.lang.String" length="2">
     <void index="0">
      <string>sync4j.server.store.SyncPersistentStore
      </string>
     </void>
     <void index="1">
      <string>sync4j.server.store.EnginePersistentStore
      </string>
     </void>
   </array>
  </void>
 </object>
</java>
```

A server JavaBean is an XML file with a particular syntax that is commonly used to set the properties of an instance of a Java class. The syntax of the server JavaBean, therefore, reflects some concepts more common in development practices. Maria is not required to know the details of the conventions used by these files, but for the sake of completeness, following is a list of the elements used, each representing a method call:

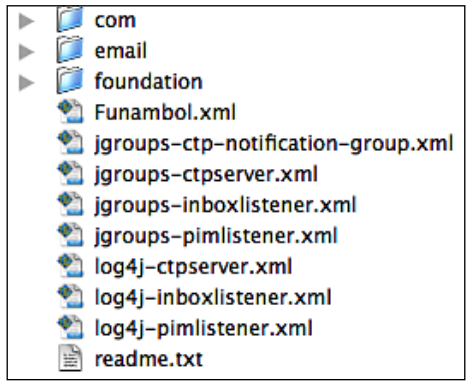
- The `object` tag denotes an expression whose value is to be used as the argument to the enclosing element.
- The `void` tag denotes a statement that will be executed, but whose result will not be used as an argument to the enclosing method.
- Elements containing other elements use them as arguments, unless they have the `void` tag.
- The name of the method is denoted by the `method` attribute.
- XML's standard `id` and `idref` attributes are used to make references to previous expressions, to deal with circularities in the object graph.
- The `class` attribute is used to specify the target of a static method or constructor explicitly, its value being the fully qualified name of the class.
- Elements with the `void` tag are executed using the outer context as the target if no target is defined by a `class` attribute.
- Java's String class is treated specially and is written `<string>Hello, world</string>`, where the characters of the string are converted to bytes using the UTF-8 character encoding.

Although all object graphs may be written using just these three tags, the following definitions are included so that common data structures can be expressed more concisely:

- The default method name is `new`.
- A reference to a Java class is written in the form `<class>javax.swing.JButton</class>`.
- Instances of the wrapper classes for Java's primitive types are written using the name of the primitive type as the tag. For example, an instance of the Integer class could be written as `<int>123</int>`. Java's reflection is internally used for the conversion between Java's primitive types and their associated wrapper classes.
- In an element representing a nullary method whose name starts with `get`, the method attribute is replaced with a `property` attribute whose value is given by removing the `get` prefix and decapitalizing the result.
- In an element representing a monadic method whose name starts with `set`, the `method` attribute is replaced with a `property` attribute whose value is given by removing the `set` prefix and decapitalizing the result.
- In an element representing a method named `get` taking one integer argument, the `method` attribute is replaced with an `index` attribute whose value is the value of the first argument.
- In an element representing a method named `set` taking two arguments, the first of which is an integer, the `method` attribute is replaced with an `index` attribute whose value is the value of the first argument.
- A reference to an array is written using the `array` tag. The `class` and `length` attributes specify the subtype of the array and its length respectively.

Configuration path

Funambol configuration files are grouped under a directory known as the **configuration path**. The name of this directory is `$FUNAMBOL_HOME/config`, whose layout is shown in the following screenshot:



At the top level, there are some important system-level configuration files such as `Funambol.xml` (that are described in more detail in the next section), the cluster configuration files (`jgroups-*`), and the logging configuration for Funambol services. In addition to those files, other files specific to particular system functionality are stored in subdirectories under the Funambol configuration path. For example, database connection configuration is stored in `com/funambol/server/db/db.xm` which, in the case of Maria's personal installation, looks like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0" class="java.beans.XMLDecoder">
 <object class="com.funambol.server.db.
               DataSourceConfiguration">
   <void method="setProperty">
    <string>url</string>
```

```
      <string>jdbc:hsqldb:hsql://localhost/funambol
      </string>
     </void>
     <void method="setProperty">
      <string>driverClassName</string>
      <string>org.hsqldb.jdbcDriver</string>
     </void>
     <void method="setProperty">
      <string>username</string>
      <string>sa</string>
     </void>
     <void method="setProperty">
      <string>password</string>
      <string></string>
     </void>
   </object>
  </java>
```

Changes to configuration files are usually picked up automatically by Funambol services. No restart or reinitialization of the services is required after a change is made. We will explicitly highlight when a configuration change requires the restart of a service in the next sections of the book.

Funambol.xml

As mentioned earlier, a particularly relevant configuration file is `Funambol.xml`, which can be found under the configuration path. This file contains important information about the Funambol server and acts as a starting point to link other configuration files of key Funambol components.

The typical contents of this file are similar to the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<java version="1.4.2" class="java.beans.XMLDecoder">
 <object class="com.funambol.server.config.
               ServerConfiguration">
   <!-- ENGINE CONFIGURATION -->
   <void property="engineConfiguration">
     <object class="com.funambol.server.config.
                   EngineConfiguration">
       <void property="officer">
         <string>
           com/funambol/server/security/
              UserProvisioningOfficer.xml
         </string>
       </void>
       <void property="serverURI">
         <string></string>
       </void>
       <void property="sessionHandler">
         <string>com.funambol.server.session.
              SyncSessionHandler
         </string>
       </void>
       <void property="strategy">
         <string>com/funambol/server/engine/Strategy.xml
         </string>
       </void>
       <void property="userManager">
         <string>com/funambol/server/admin/
             DBUserManager.xml
         </string>
       </void>
       <void property="minMaxMsgSize">
         <long>1800</long>
       </void>
       <void property="deviceInventory">
         <string>com/funambol/server/inventory/
                     PSDeviceInventory.xml
         </string>
```

```
      </void>
      <void property="dataTransformerManager">
        <string>
          com/funambol/server/engine/transformer/
          DataTransformerManager.xml
        </string>
      </void>
      <void property="checkForUpdates">
        <boolean>true</boolean>
      </void>
    </object>
  </void>
  <!-- SERVER INFO -->
    <void property="serverInfo">
      <object class="com.funambol.framework.core.
                     DevInf">
        <object class="com.funambol.framework.core.
                       VerDTD">
          <!-- verDTD  -->
          <string>1.2</string>
        </object>
        <!-- man ->
        <string>Funambol</string>
        <!-- mod -->
        <string>DS Server</string>
        <!-- oem -->
        <string>-</string>
        <!-- fwV -->
        <string>-</string>
        <!-- swV -->
        <string>7.1.0</string>
        <!-- hwV -->
        <string>-</string>
        <!-- devID -->
        <string>funambol</string>
        <!-- devTyp -->
        <string>server</string>
        <!-- utc -->
        <boolean>true</boolean>
        <!-- supportLargeObjs -->
        <boolean>true</boolean>
        <!-- supportNumberOfChanges -->
```

```
            <boolean>true</boolean>
            <array class="com.funambol.framework.core.
                          DataStore" length="0"/>
            <array class="com.funambol.framework.core.Ext"
                   length="0"/>
          </object>
        </void>
    </object>
  </java>
```

It is likely that Maria will not be able to understand this file at first. Fortunately, it is unlikely that she will need to directly modify it. For now, it is sufficient to note that it contains the configuration of the synchronization engine in the section `engineConfiguration` and server capabilities in the section `serverInfo`.

Summary

In this chapter, Maria learned how to configure the Data Synchronization Service. She started with how to start the service and how to connect to it with the Funambol administration tool. She then learned how Funambol logging works and can be configured and how to configure Funambol to access a database on a separate machine instead of the local database embedded in a standard installation. Maria now also understands how to use an Apache web server as an HTTP or HTTPS frontend in front of the Funambol Data Synchronization Service.

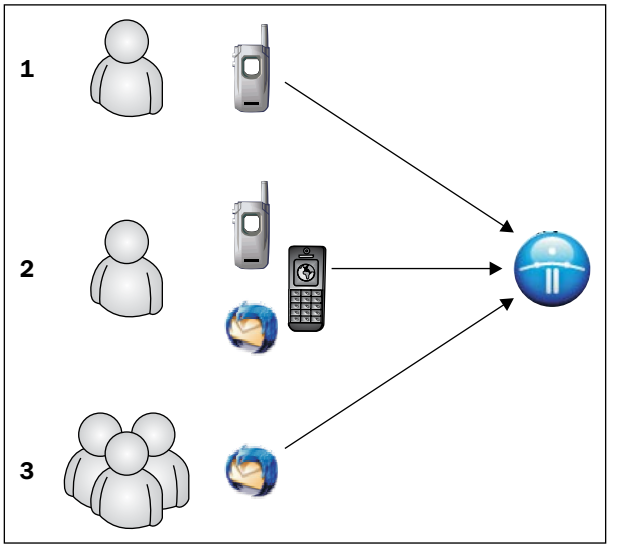
In addition, Maria learned more about the Funambol configuration architecture so that she feels more comfortable in effectively managing a Funambol deployment.

4
Administrating Users and Devices

In the previous chapter, Maria learned how to deploy the Funambol Data Synchronization Service by making the configuration changes that are necessary for an enterprise installation. As an administrator of the system, she needs to administer users and devices. In this chapter, Maria will learn how to use the Funambol Administration Tool to see which users are registered to the server, the devices they are using, and how to enable or disable user access to the system.

Users, devices, and principals

In a heterogeneous environment such as Maria's company, users have different needs. Some will use just their own mobile phones and backup their contacts and calendar only, while others will keep their mobile data synchronized with their desktop or a portable device such as the iPod. Another possible scenario is that the same desktop computer could be shared by different users. Maria may not necessarily encounter all of these cases, but Funambol supports all these possibilities.



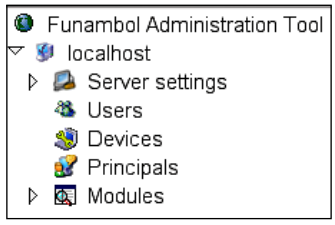
As illustrated in this figure, the first case represents a 1:1 relationship between a Funambol user and her device. In the second case, one person uses many devices; this is a 1:N relationship. The last case is perhaps more uncommon and represents an N:1 relationship between multiple users and a single device.

In Funambol, each association between a specific user and a particular device she owns is called **principal**. Maria, for instance, wants to keep in sync her Thunderbird address book with her mobile phone. To support this, Funambol needs two principals each representing a maria—device association so that:

```
<maria, thunderbird> - <maria, syncml phone>
```

With these associations the server is able to determine the status of the synchronization for each device or application used by a given user, at each time, and consequently determine which information has to be sent during a synchronization session.

Users, devices, and principals are managed by means of the **Funambol Administration Tool**. After logging in to the server and expanding the server node, Maria can access the management of these entities by double-clicking on the corresponding items as illustrated in the following screenshot:



Auto and manual provisioning

With Funambol, Maria can apply different policies while adding users to the system. Depending on her company's security requirements, she can opt for automatic provisioning of users or for a manual and more controlled provisioning method.

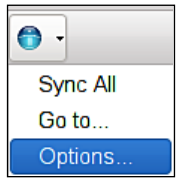
Provisioning is the term used to represent the process of adding all information to the system, required to identify a user and allow her to carry on the synchronization.

Auto users provisioning

By default, Funambol performs automatic provisioning of new users. With this method, when a user synchronizes with the server for the first time, the username, password, device, and principal information is automatically added to the Funambol database. Of course, this process only occurs the first time a user launches a synchronization. If the user is already in the Funambol repository with the same username and is synchronizing with a device used in the past, no information is changed. On the other hand, if the user synchronizes with a new device, a new user-device association will be created. In any case, the password will never be automatically updated after the user has been created.

Maria has already experienced this feature when she synchronized for the first time in Chapter 1. To see auto-provisioning at work, Maria will now synchronize her address book with her username instead of the username "guest".

In Thunderbird, she opens the configuration panel of the Funambol Mozilla Sync Client by using the Funambol button bar shown in the following screenshot:



She then types "maria" in the **Username** field and a password of her choice in the **Password** field.

The synchronization is triggered by clicking on the synchronization icon, and all necessary data is automatically stored in the Funambol database.

Maria can check this by opening the Funambol Administration Tool and double-clicking on **Users**. This brings up the **Search Users** panel in the right pane of the tool. Clicking the **Search** button will bring up a list of all the users in the system. She will notice there is a new user **maria**, as illustrated in the following screenshot:

Username	First Name	Last Name	E-mail	Roles
admin	admin	admin	admin@funa...	Administrator
guest	guest	guest	guest@funa...	User
maria				User

In a similar way, when searching the **Devices** section, Maria will find the entry representing the Mozilla Sync Client (identified by an ID starting with "fmz-") and the principal that associates the device with the user "maria".

This method of user provisioning is convenient for demo purposes or for an open service. In other cases a manual approach may be more appropriate, as described in the next section.

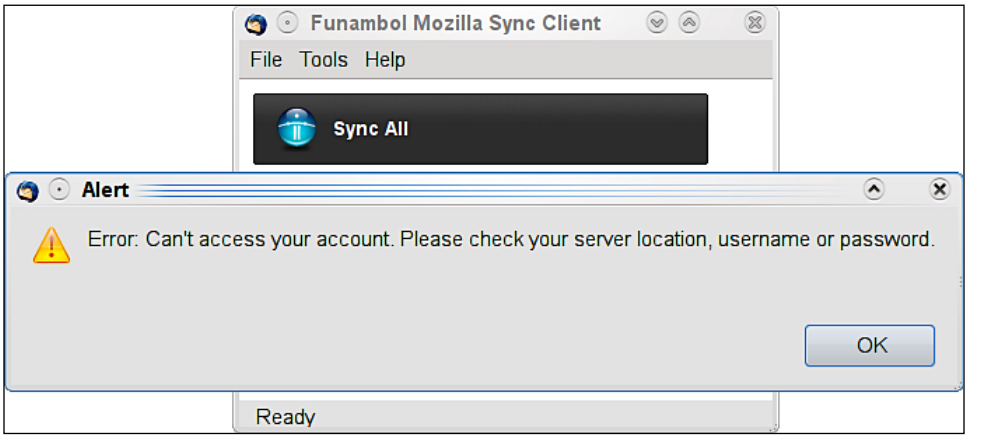
Manual user provisioning

Maria will operate the Funambol service in a controlled environment, where she will provide access to the synchronization service to a selected group of users. She does not want users to be provisioned automatically. Therefore, Maria will manually add users to the system and also add the devices that they are allowed to synchronize.

To do this, Funambol must be configured to use a different "officer", the component that authenticates users and which, in auto-provisioning mode, adds the user information to the database. Maria can do this by opening the Funambol Administration Tool and displaying the server information. As seen in the previous chapter, the default officer used by Funambol is `com/funambol/server/security/UserProvisioningOfficer.xml`. To enable manual user provisioning, Maria just needs to change it to `com/funambol/server/security/DBOfficer.xml` and click on **Save**. The change will immediately take effect.

To check that the change provides the desired behavior, Maria performs two simple tests:

- First, she synchronizes again with the Funambol Mozilla Sync Client. This synchronization completes correctly because the user "maria" was previously provisioned (automatically).
- The second test is to change the username to a non-existing user in the configuration panel of the Mozilla Sync Client and attempt to synchronize. This time the synchronization will terminate with an authentication error, as shown in the following screenshot:



Maria can now add more users and allow them to synchronize using the Funambol Administration Tool as described in the following section.

Adding a new user

Maria can use the Funambol Administration Tool to:

- Register the new user
- Register the device the user will be allowed to synchronize with
- Create a principal that associates the two

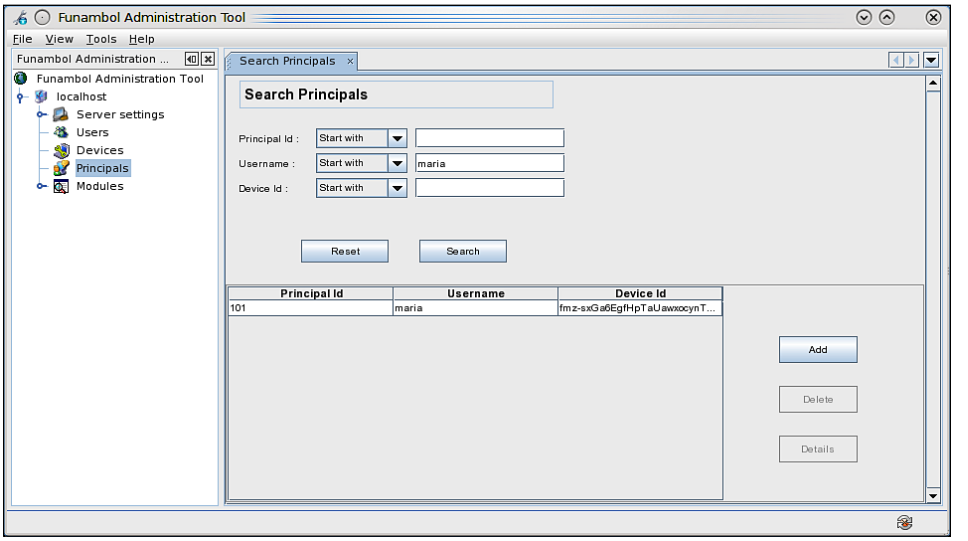
To add a new user, Maria needs the following information:

- Username
- Password
- First and last name
- E-mail
- Device ID

The method to acquire the device ID differs between devices. Funambol Sync Clients display the device ID in their advanced options, but other devices, for instance SyncML phones, may not provide such information. Most mobile phones use their **International Mobile Equipment Identity (IMEI)**, (see `http://en.wikipedia.org/wiki/Imei`) as the device ID or a string in the form IMEI:xxxxxxxxxxxx. However, they do not offer an easy way to determine which ID they will use for the synchronization client.

One easy way to grab the device ID is to use an installation of Funambol with automatic provisioning enabled. Performing a test synchronization with such a server will cause the device IDs to be automatically inserted in the database and make them accessible through the Funambol Administration Tool.

For instance, to learn the device ID of her Thunderbird Sync Client, Maria can perform a test synchronization as explained earlier, and then connect into the test server with the **Funambol Administration Tool**. After double-clicking on **Principals**, the **Search Principals** panel is displayed, as illustrated in the following screenshot:



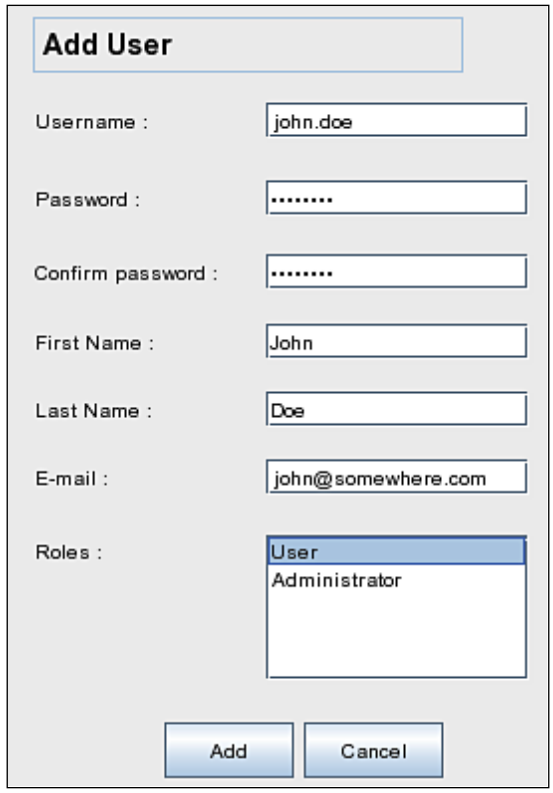
Searching for the username **maria**, she finds that there is a new principal and that the **Device ID** of her Mozilla Sync Client is **fmz-sxGa6EgfHpTaUawxocynTg==**.

In the next steps, Maria will manually add a new user and device with the following characteristics:

- **Username: john.doe**
- **Password: 1234567890**
- **First name: John**
- **Last name: Doe**

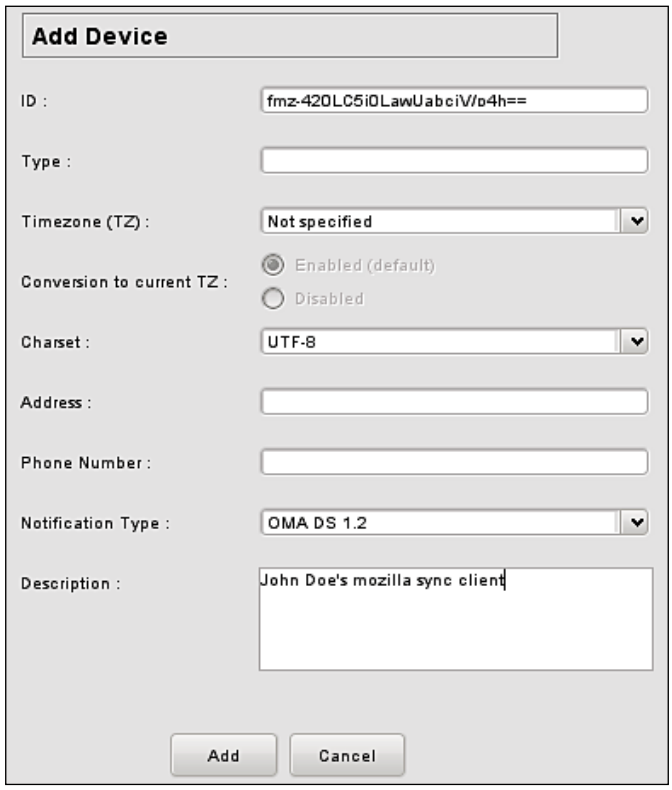
- **E-mail: john.doe@somewhere.com**
- **Device ID: fmz-420LC5i0LawUabciV/p4h==** (supposing this is John's Mozilla Sync Client's ID)

After logging in to the Funambol Administration Tool (this time connecting to the real server that does not have auto-provisioning enabled), Maria right-clicks on **Users** and selects **Add User**. The **Add User** panel appears in the right pane. Maria inserts the information as shown in the following screenshot, and clicks **Add**.

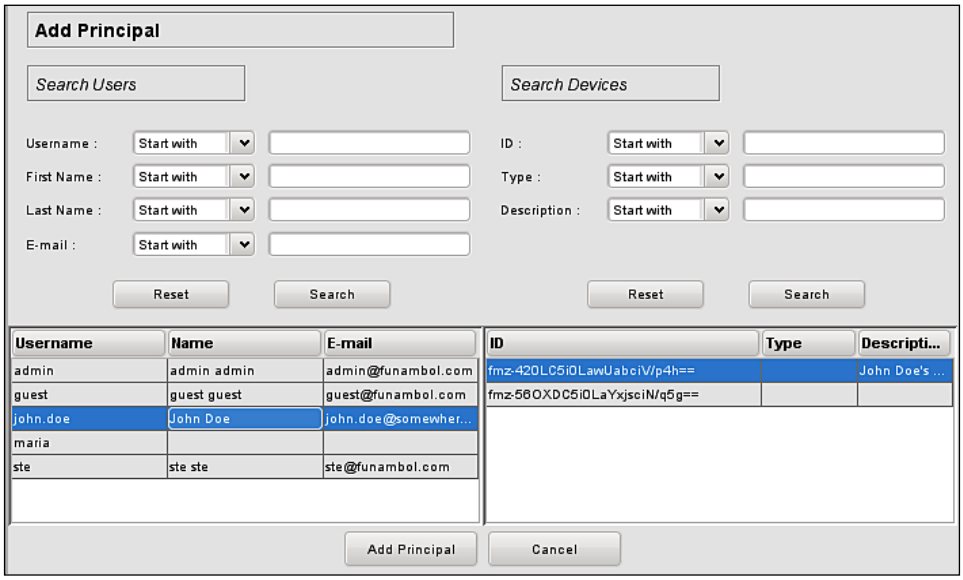


Note that Maria selected the role **User** in the **Roles** list. This gives access to the synchronization service only. In the same way, she can create other administrators—other people that can use the Funambol Administration Tool—by selecting the **Administrator** role.

Maria now registers the device. Right-clicking on **Devices**, she selects **Add Device** and enters the information as displayed in the following screenshot:



Finally, Maria creates the principal—an association between the newly registered user and the device. By right-clicking on **Principals** and selecting **New Principal**, the **Add Principal** panel appears in the right pane of the Funambol Administration Tool. This panel has two search sections—one to search for users and the other to search for devices. Clicking the **Search** button in both sections displays all users and devices registered. Maria selects the desired user and device and clicks **Add Principal**, as shown in the following screenshot:



Done! The user **john.doe** is now registered in the system and ready to synchronize his Thunderbird data with the Funambol server.

Other administration tasks

In this section, additional management tasks that Maria may find helpful are described. These are:

- Searching users, devices, and principals
- Deleting users, devices, and principals
- Inspecting device capabilities

Searching users, devices, and principals

Maria has already used the Administration Tool's functions to search users, devices, and principals. Search panels provide more functionality than simply returning a list of all records. Each search panel presents a number of fields, a set of search functions for each field, and an entry for each field. For example, in the **Search Users** panel, it is possible to filter the fields **Username**, **First Name**, **Last Name**, and **E-mail**. For each field, the available search functions are **start with, end with**, **contain**, and **exactly**. The server returns the records that satisfy all specified search criteria. For example, with the information shown in the following screenshot, searching for **E-mail** fields containing "somewhere.com" returns both **maria** and **john.doe**.

Username	First Name	Last Name	E-mail	Roles
admin	admin	admin	admin@funambol.com	Administrator
guest	guest	guest	guest@funambol.com	User
john.doe	John	Doe	john.doe@somewhere.com	User
maria	Maria	White	maria.white@somewhere.com	User

However, a search for **First Name** containing "john" and **E-mail** containing "somewhere.com", returns **john.doe** only.

The search forms for devices and principals work in exactly the same way.

Note that for performance reasons, the server returns 100 results max per query. It is recommended that Maria makes selective searches once there are many entries in the Funambol database.

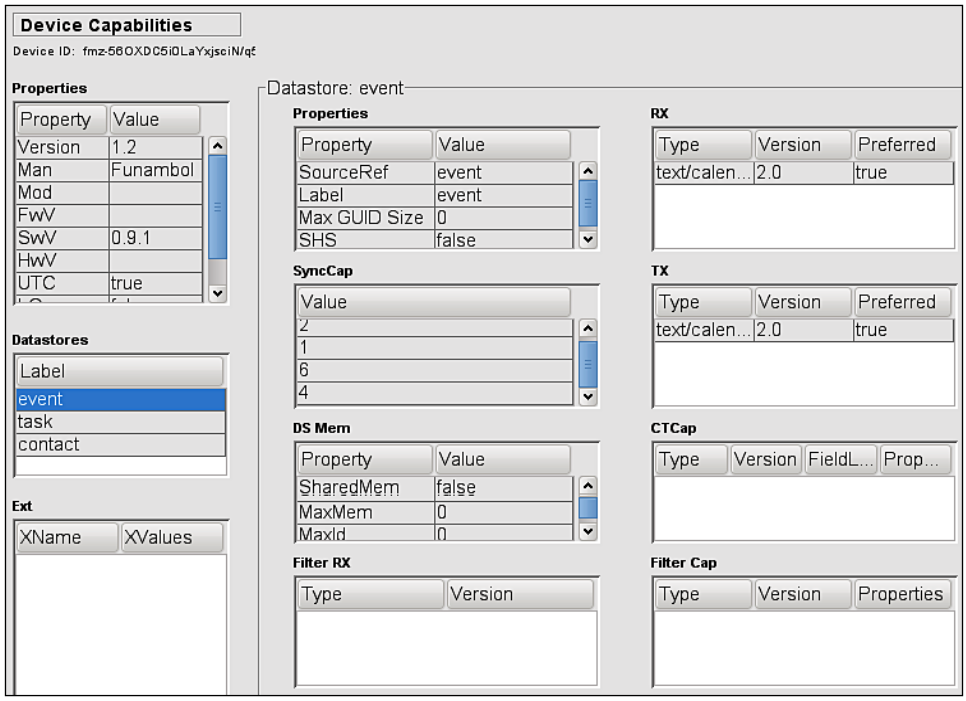
Deleting users, devices, and principals

Deleting existing entries is straightforward. After finding the entry to delete from the **Search** panel, Maria just needs to select it and click the **Delete** button. She will be asked to confirm the deletion and then the item will be deleted from the system.

Inspecting device capabilities

In the **Device Search** panel, there is a button that is not present in any other panel—the **Capabilities** button. By clicking this button, Maria can inspect the capabilities of the selected device. **Device capabilities** provides a set of information about the capabilities of the SyncML client on the device. This includes information such as the SyncML version supported, which data types the device is able to synchronize, which fields, and so on.

For example, the capabilities of the Funambol Mozilla Sync Client are reported in the following screenshot:



A brief description of the information displayed is given in the following table:

Field	Description
Properties	Information about the client implementation including the protocol version supported by the client, the hardware, firmware, and software version of the device, whether the device uses UTC dates, and other protocol capabilities.
Datastores	Data types synchronized by the device. When a data type is selected, the information about the selected datastore is displayed in the right-hand side part of the capabilities panel.

Field	Description
Ext	Device specific extensions.
Datastore properties	Identification information of the datastore.
Datastore SyncCap	Synchronization types supported by the device.
Datastore DS Mem	Datastore memory capabilities.
Datastore RX	Accepted MIME type for items received by the server.
Datastore TX	MIME type used to send items to the server from this datastore.
CT Cap	Field information about the supported types.
Filter RX	Filtering capabilities in receiving items for this datastore.
Filter Cap	Additional filtering capabilities.

Summary

In this chapter Maria learned how to manage user access to the synchronization service. She learned how to use the Funambol Administration Tool to configure Funambol to restrict access to specific users and how to add new users. She also learned more about other management tasks that she may be required to perform periodically.

5
Funambol E-mail

So far Maria has primarily learned and experimented with Funambol PIM synchronization. It is now time to introduce one of the most exciting capabilities of Funambol: Mobile e-mail.

One of the biggest advantages of the Funambol platform is that with the same system, Maria can provide e-mail and PIM sync on many mobile platforms, including:

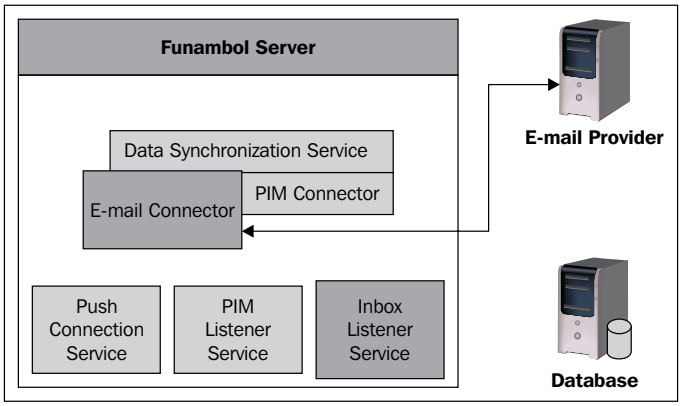
- Windows Mobile PDA and phones
- Symbian smartphones
- BlackBerry phones
- Java-enabled phones
- Apple iPhone

In this chapter, Maria will set up Funambol to connect to the company e-mail server, in order to enable her users to receive e-mail on their mobile phones.

E-mail Connector

Maria became familiar with the E-mail Connector in Chapter 2. The E-mail Connector allows Funambol to connect to any IMAP and POP e-mail server to enable mobile devices to receive corporate or personal e-mail.

The part of the Funambol architecture involved with this functionality is illustrated in the following figure:



As described in Chapter 2, the E-mail Connector is a container of many things, the most important ones being:

- The e-mail server extension (represented in the figure by the **E-mail Connector** block): This is the extension of the Funambol Data Synchronization Service that allows e-mail synchronization through the connection to the e-mail provider.
- The Inbox Listener Service: This is the service that detects new e-mail in the user inbox and notifies the user's devices.

When the Funambol DS Service receives sync requests for e-mail, the request calls the E-mail Connector, which downloads new messages from the e-mail server and makes them available to the DS Service, which in turn delivers them to the device.

When a user receives a new e-mail, the new message is detected by the Inbox Listener Service that notifies the user's device to start a new sync.

When the E-mail Connector is set up and activated, e-mail can be synced with an e-mail provider if it supports one of the two popular e-mail protocols—POP3 or IMAP v4 for incoming e-mail and the SMTP protocol for outgoing e-mail delivery.

Please note that the Funambol server does not store user e-mail locally. For privacy and security reasons, e-mail is stored in the e-mail store of the **E-mail Provider**. The server constructs a snapshot of each user's inbox in the local database to speed up the process of discovering new e-mails without connecting to the e-mail server. Basically, this e-mail cache contains the ID of the messages and their received date and time.

The Funambol service responsible for populating and updating the user inbox cache is the Inbox Listener Service. This service checks each user inbox on a regular basis (that is, every 15 minutes) and updates the inbox cache, adding new messages and deleting the messages that are removed from the inbox (for example, when a desktop client downloaded them or the user moved the messages to a different folder).

Another important aspect to consider with mobile e-mail is that many devices have limited capabilities and resources. Further, the latency of retrieving a large inbox can be unacceptable for mobile users, who need the device to be always functional when they are away from their computer. For this reason, Funambol limits the maximum number of e-mails that Maria can download on her mobile so that she is never inconvenienced by having too many e-mails in her mobile e-mail inbox. This value can be customized in the server settings (see section *E-mail account setup*).

In the following sections, Maria will learn how to set up Funambol to work with the corporate e-mail provider and how she can provide Funambol mobile e-mail for her users.

Setting up Funambol mobile e-mail

The Funambol E-mail Connector is part of a default installation of Funambol so Maria does not need to install any additional packages to use it. The following sections describe what Maria needs to do to set up Funambol to connect to her corporate e-mail server.

E-mail Provider

The only thing that Maria needs to make sure about the corporate E-mail Provider is that it supports POP/IMAP and SMTP access from the network where Funambol is installed.

It is not necessary that the firewall is open to mobile devices. Devices will keep using SyncML as the transport protocol, while the Funambol server connects to the e-mail server when required.

Also, the same e-mail server does not need to provide both POP (or IMAP) and SMTP. Funambol can be configured to use two different servers for incoming and outgoing messages.

Funambol authentication with e-mail

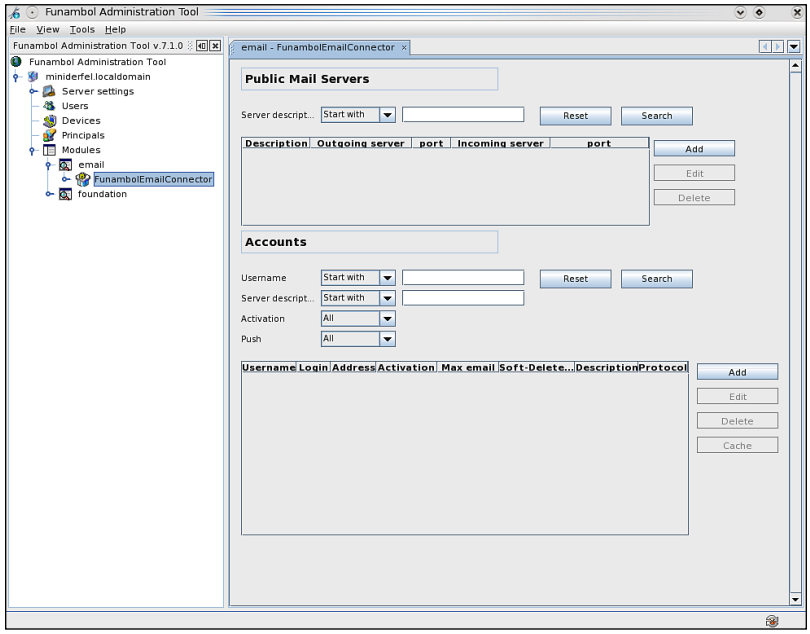
One of Maria's security concerns is the distribution and provisioning of e-mail account information on the mobile phones. She does not like the fact that e-mail account information is sent over a channel that she can only partially control.

This is a common concern of IT administrators. Funambol addresses this issue by not storing e-mail credentials on the device. The device (or any other SyncML client) is provisioned with Funambol credentials. In the previous sections, Maria was able to create new accounts so that users could use the PIM synchronization service, and in doing so, she needed to provide new usernames and passwords. This is still valid for e-mail users.

What Maria needs to do now is to configure the E-mail Connector and add the e-mail account of the users she wants to enable for mobile e-mail. These topics are covered in detail in the following sections.

E-mail account setup

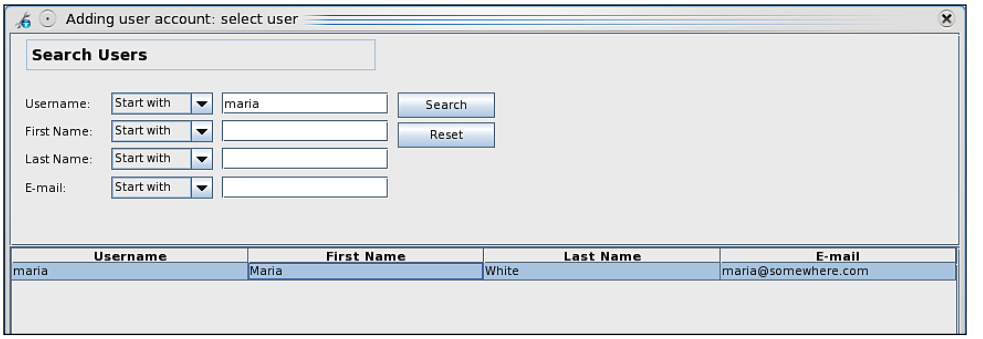
To add a user e-mail account to the synchronization service, Maria can use the Funambol Administration Tool, expanding the **Modules | email | FunambolEmailConnector** node and double-clicking the connector. This opens the connector administration panel, as shown in the following screenshot:



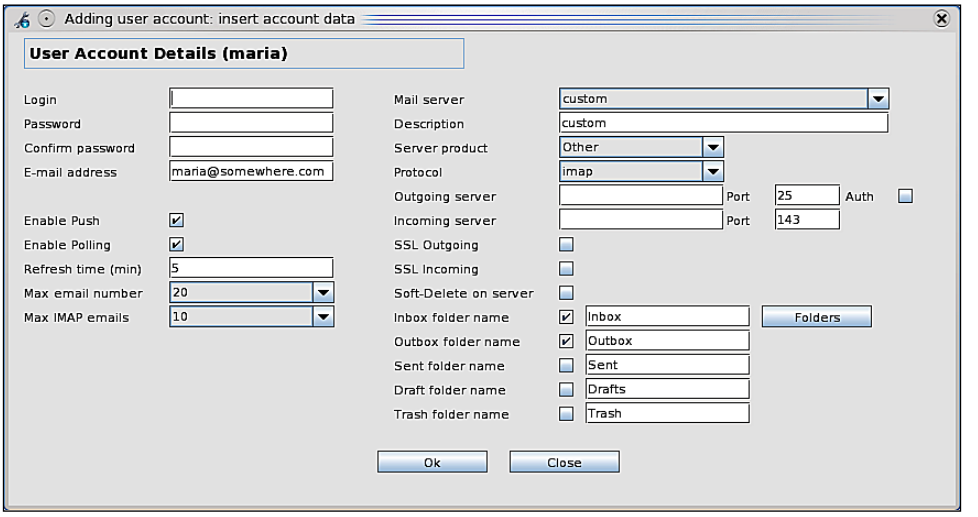
There are two sections: **Public Mail Servers** and **Accounts**.

Maria needs to add new accounts. Let's start with her account first.

Clicking the **Add** button in the **Accounts** section opens up a new search window so that she can search which Funambol user to attach to the e-mail account. Typing **maria** in the **Username** field and clicking **Search**, will show you the result as shown in the following screenshot:



Double-clicking the desired account displays a form for maria's account details as shown in the following screenshot:



Each field is explained as follows:

- **Login**, **Password**, **Confirm password**, and **E-mail address**

  As the labels of the fields describe, these are the e-mail account credentials and e-mail address. These are credentials used to access the e-mail service, not the ones to access the Funambol synchronization service.

- **Enable Polling**

  This setting enables or disables the functionality of the Inbox Listener Service to check for updates on this account's inbox. When disabled, the account inbox won't be scanned for new/updated/deleted e-mail. This disables e-mail synchronization completely.

- **Enable Push**

  This setting enables or disables the push functionality. When disabled, the user will not be notified of new e-mails. If **Enable Polling** checkbox is active, the Inbox Listener Service keeps updating this account's e-mail cache anyway. In this case Maria can still download e-mail by manually starting the synchronization from the client.

- **Refresh time(min)**

  This setting specifies how frequently the Inbox Listener Service checks for updates on this account's inbox. The value is expressed in minutes. The shorter this period, the more often new e-mail is detected and therefore the closer the user experience is to real time. However, the smaller this number, the heavier the load on the Inbox Listener Service and the e-mail provider. When you have only a few users, this is not too relevant, but it is something to consider when planning a major deployment.

  For the tests of the following section, Maria will set this value to 1 minute.

- **Max email number**

  This is the maximum number of e-mails tracked in the account's cache. Maria can adjust this value based on each device that needs to be supported. For a low-end device, this value should not be greater than 30 or 50. For smartphones and more powerful devices, this value can be a few hundred.

  In general, mobile users are not interested in having all of their e-mails on their phone but rather the most recent messages, which usually require immediate attention when they are away. A good rule of thumb to select the correct number is the number of new messages a user receives daily.

- **Mail server**, **Server product**, and **Description**

  Certain e-mail servers, in particular public servers such as Yahoo! and Gmail have particular implementations of the POP/IMAP protocols that require special processing. The **Mail server** settings allow Maria to specify if the account is one of those servers. Supported server types include:

  - Custom (any POP/IMAP service)
  - Gmail
  - Google Mail (Google hosted mail)
  - Yahoo! (premium account)
  - AOL
  - Hotmail (premium account)

  For now, Maria will configure her corporate account. She will select **Custom** as **Mail server** and **Other** as **Server product**. She can also put the server name in the **Description** field. If one of the above public servers is selected, many of the following settings will be automatically set to the correct server values and cannot be modified.

- **Protocol**

  This setting specifies which protocol the E-mail Connector shall use to access Maria's corporate account. The two possible choices are POP and IMAP.

- **Outgoing server**, **Port**, **Auth**, and **SSL Outgoing**

  These fields allow Maria to specify which SMTP server to use and if it requires any settings different from the standard behavior (in particular, the SMTP port and whether the e-mail server requires authentication).

  Funambol currently supports authentication through the e-mail credentials (e-mail address and password).

  If Maria's corporate e-mail server requires SSL/TLS, she needs to select **SSL Outgoing**.

- **Incoming server**, **Port**, and **SSL Incoming**

  With these settings, Maria can set the hostname and port of the corporate e-mail provider. If the mail server requires SSL/TLS access, the setting **SSL Incoming** must be checked.

- **Soft-Delete on server**

  Soft-delete is when the user deletes a message on the device, but the message is not physically deleted on the e-mail server. This option is particularly helpful to avoid a message being accidently deleted or an e-mail being accidently removed by a user who is cleaning his/her mobile inbox(screen space is a precious resource on many mobile phones). When this option is checked, messages deleted from the mobile client will not be deleted on the e-mail server so that they are still accessible from a desktop or web-based e-mail client.

- **Inbox**, **Outbox**, **Sent**, **Draft**, and **Trash folder name**

  When using IMAP, the top-level folder names for Inbox, Outbox, Sent, Draft, and Trash may change based on the server implementation or even the user language. These settings (enabled only when the selected protocol is IMAP) allow Maria to map the folder names on the server to the logical Inbox, Outbox, Sent, Draft, and Trash. She can also click the **Folders** button, as a result the Funambol Administration Tool tries to detect the names of the folders by retrieving them from the mail server.

Inbox Listener Service

The default installation of Funambol is already set up to start the Inbox Listener Service when launching the following command:

```
$FUNAMBOL_HOME/bin/funambol start
```

However, in a more scalable environment, it is recommended to install this service on a separate box (potentially more than one, depending on the number of e-mail users that the system must serve). The easiest way to do this is to just install Funambol on the dedicated system and start the Inbox Listener Service only.

This is done by launching the following command:

```
$FUNAMBOL_HOME/bin/inbox-listener start
```

If the Inbox Listener Service is not running, the user's e-mail cache will not be updated and no new e-mail will be detected.

Maria can check that the service is running, by simply issuing the `ps` command and checking that the output is similar to the following:

```
ps -ef |grep java
maria      25320     1  0 13:34 pts/2    00:00:15 /
opt/Funambol/tools/jre-1.5.0/jre/bin/java -Dfile.
encoding=UTF-8 -Dfunambol.home=/opt/Funambol -
Dfunambol.pushlistener.config.bean=com/funambol/email/
inboxlistener/InboxListenerConfiguration.xml -Djava.
library.path=/opt/Funambol/inbox-listener/lib/linux -
Djavax.net.ssl.trustStore=/opt/Funambol/inbox-listener/
lib/security/cacerts -Djava.net.preferIPv4Stack=true
-Xmx256M -Dcom.sun.management.jmxremote -Dcom.sun.
management.jmxremote.port=4101 -Dcom.sun.management.
jmxremote.ssl=false -Dcom.sun.management.jmxremote.
authenticate=false
```

Inbox Listener Service Configuration

Like explained in chapter 2, once the Inbox Listener Service detects a new e-mail, it tells the Data Synchronization Service to notify the device to start a new synchronization. This is done with an HTTP remote procedure call, therefore, if the Inbox Listener and Data Synchronization services are on different hosts, the Inbox Listener must be configured with the host name of the Data Synchronization Service.

Maria can do it editing the configuration file `$FUNAMBOL_HOME/com/funambol/email/inboxlistener/InboxListenerConfiguration.xml` and setting the appropriate values in the `serverInformation` section like in the example below:

```
<void property="serverInformation">
 <object class="com.funambol.server.admin.ws.client.
               ServerInformation">
  <void property="url">
   <string>http://server:port/funambol/services/
                admin</string>
  </void>
  <void property="username">
   <string>admin</string>
  </void>
  <void property="password">
   <string>password</string>
  </void>
 </object>
</void>
```

Note that after this change the service must be restarted.

Inbox Listener troubleshooting

To troubleshoot Inbox Listener issues, Maria can access a lot of information parsing the log file.

For technical and historical reasons, at the time of this writing, the Inbox Listener Service's log facility cannot be configured from the Funambol Administration Tool as seen in Chapter 3. In order to do so, Maria needs to edit the file `$FUNAMBOL_HOME/config/log4j-inboxlistener.xml`. For example, to enable full logging, she needs to turn the level of the funambol logger to `ALL`.

```
<logger name="funambol">
    <level value="ALL"/>
</logger>
```

The Inbox Listener log file is located, by default, under `$FUNAMBOL_HOME/logs/inbox-listener`.

Mobile e-mail at work

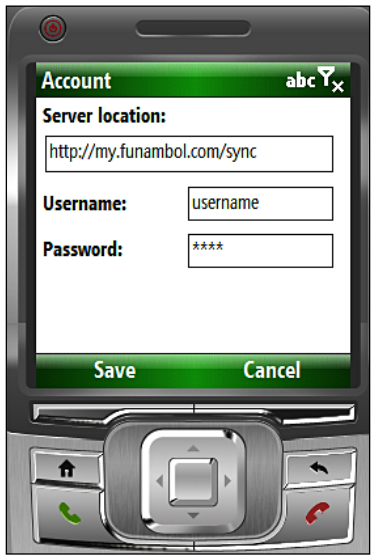
One of the most widely used phones for mobile e-mail are phones running Windows Mobile; therefore, this is a platform Maria will have to support. Funambol fully supports this platform, extending the Windows Mobile native e-mail client to support SyncML and Funambol mobile e-mail.

As Windows Mobile does not natively support SyncML, Maria needs to download the Funambol Windows Mobile Sync Client from the following URLs:

- `http://www.funambol.com/opensource/download.php?file_id=funambol-smartphone-sync-client-7.2.2.exe&_=d` (for Windows Mobile smartphone)
- `http://www.funambol.com/opensource/download.php?file_id=funambol-pocketpc-sync-client-7.2.2.exe&_=d` (for Windows Mobile Pocket PC)

Like any other Windows Mobile applications, these are executable files that need to be run on a desktop PC and the installation will be performed by Microsoft ActiveSync.

Once installed on the mobile phone, Maria can run Funambol by clicking the Funambol icon. The first time the application is launched, it asks for the Funambol credentials, as shown in the following image:

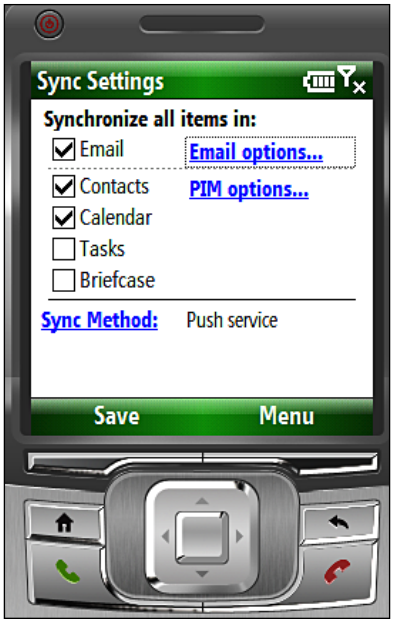


Maria fills in her Funambol Server location and credentials (not her e-mail account credentials) and presses **Save**. After a short while, the device will start downloading the messages that she can access from the Funambol account created by the Funambol installation program in Pocket Outlook. The inbox will look similar to the following image:

To see mobile e-mail at work, Maria just needs to send an e-mail to the e-mail account she set up earlier. In less than a minute, her mobile device will be notified that there are new messages and the synchronization will automatically start (unless the client is configured differently).

Mobile e-mail client configuration

There are a number of settings that Maria can set on her mobile phone to change how mobile e-mail works. These settings are accessible from the Funambol application by clicking on **Menu** | **Settings**.



There are two groups of settings that are important for mobile e-mail: **E-mail options...** and **Sync Method**.

From the **Email options** panel, Maria can choose which e-mails to download (all e-mails, today's e-mails, or e-mails received from the last X days), the size of the e-mail to download first (then the remaining part can be downloaded on demand), and if she also wants to download attachments. In the **advanced options**, she can also choose to use a different "From" display name and e-mail address.

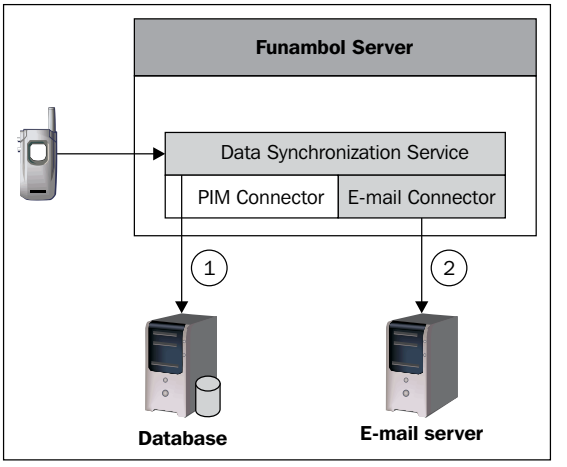
From the push method panel, Maria can choose how to download e-mail automatically using the push service on a regular basis, with either a scheduled sync or only manually upon request (from the Funambol Windows Mobile Sync Client interface or the PocketOutlook send and receive command).

Funambol supports many mobile phones for mobile e-mail. The previous description is only for Windows Mobile phones. The manner in which Funambol supports other devices depends on the phone. In some cases, Funambol uses the phone's native e-mail client, such as with Windows Mobile. In other cases, Funambol provides its own mobile e-mail client that is downloaded onto the device. Other phones and options are covered in Chapter 7.

Improving Funambol e-mail authentication

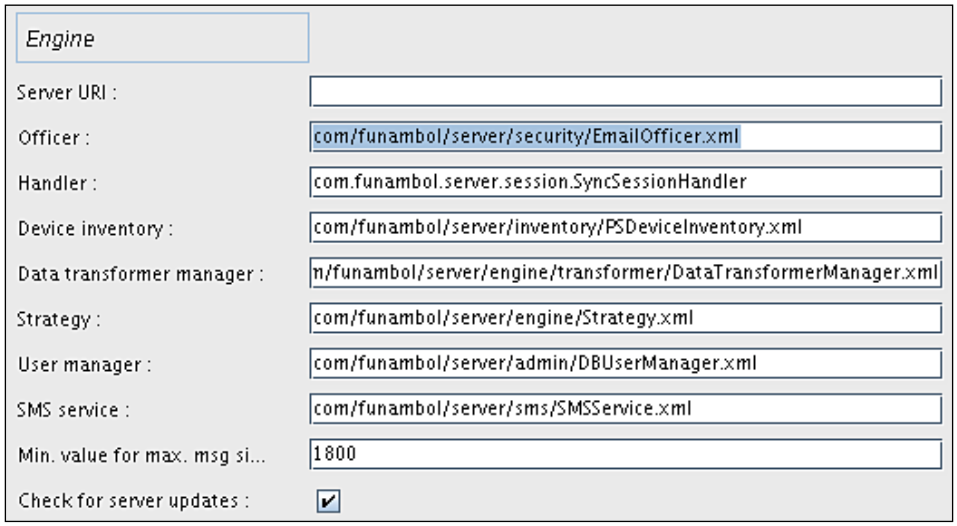
In the section *E-mail account setup*, Maria saw how to create and configure an e-mail account so that Funambol is able to access it. A piece of information she provided is the e-mail username and password. These credentials are used by Funambol to log into the e-mail server and are of course different from the synchronization credentials.

This is illustrated in the following figure, where the Funambol server first authenticates the synchronization session, and then the E-mail Connector connects to the e-mail server.



With this setup, if the e-mail server does not authenticate the e-mail credentials, the access to the e-mail server would fail and Maria would see a synchronization error on her client side. What she would have expected, instead, is an authorization failure, with a message that tells her to check username and password. This would make it easier for an administrator to check which username and password are wrong.

To do so, Maria needs to adjust the server's configuration by changing the Funambol **Officer** to `EmailOfficer.xml` as illustrated in the next screenshot, from the **Server settings** node of the Funambol Administration Tool.



Detecting authentication problems

Authentication problems can be easily detected by inspecting the server log found under `$FUNAMBOL_HOME/log/ds-server/ds-server.log` (unless Maria configured the logging differently).

The following sections show how the server log looks in the case of:

- Successful authentication
- Funambol authentication failure
- E-mail provider authentication failure

Successful authentication

If the synchronization of Funambol e-mail from the mobile phone terminates successfully, it means that the username and password have been validated by both Funambol and the e-mail provider. In this case, the server log looks like the following:

```
[2009-07-03 15:31:03,299] [funambol.transport.http]
[INFO] [8CEEF28A7322E8BC45CAC7E6F9E93A02] [] [] []
Handling incoming request
[2009-07-03 15:31:03,300] [funambol.transport.http]
[INFO] [8CEEF28A7322E8BC45CAC7E6F9E93A02] [] [] []
Request URL: http://192.168.0.74:8080/funambol/ds
[2009-07-03 15:31:03,300] [funambol.transport.http]
[INFO] [8CEEF28A7322E8BC45CAC7E6F9E93A02] [] [] []
Requested sessionId: null
[2009-07-03 15:31:03,962] [funambol.handler] [INFO]
[8CEEF28A7322E8BC45CAC7E6F9E93A02] [fwm-35313801010967P7]
[maria] [] maria/fwm-35313801010967P7 logged in.
[2009-07-03 15:31:04,340] [funambol.transport.http]
[INFO] [8CEEF28A7322E8BC45CAC7E6F9E93A02] [fwm-
35313801010967P7] [maria] [] Request processed.
```

Funambol authentication failure

The authentication credentials that Funambol checks first are the Funambol server credentials. If the user is not authenticated, the server log is similar to the following:

```
[2009-07-03 16:05:38,115] [funambol.transport.http]
[INFO] [C377D015453C8B3804CBE4EF4C5824D7] [] [] []
Handling incoming request
```

```
[2009-07-03 16:05:38,148] [funambol.transport.http]
[INFO] [C377D015453C8B3804CBE4EF4C5824D7] [] [] []
Request URL: http://192.168.0.74:8080/funambol/ds
[2009-07-03 16:05:38,148] [funambol.transport.http]
[INFO] [C377D015453C8B3804CBE4EF4C5824D7] [] [] []
Requested sessionId: null
[2009-07-03 16:05:38,583] [funambol.handler] [INFO] [C377
D015453C8B3804CBE4EF4C5824D7] [fwm-35313801010967P7] []
[] User not authenticated
[2009-07-03 16:05:38,583] [funambol.handler] [INFO] [C37
7D015453C8B3804CBE4EF4C5824D7] [fwm-35313801010967P7] []
[] Authentication failed for device fwm-35313801010967P7.
Make sure that the client used correct username and
password and that there is a principal associating the
user  to the device.
[2009-07-03 16:05:38,666] [funambol.transport.http]
[INFO] [C377D015453C8B3804CBE4EF4C5824D7] [fwm-
35313801010967P7] [] [] Request processed.
```

In this case, the user will also be notified of the error in the Funambol client.

E-mail provider authentication failure

In this case, Funambol credentials are correctly verified, but the E-mail Connector fails to authenticate the e-mail user against the e-mail provider. The server log will be similar to the following:

```
[2009-07-03 16:35:27,248] [funambol.transport.http]
[INFO] [D9044872013116F89220CDEEC2AB9D9A] [] [] []
Handling incoming request
[2009-07-03 16:35:27,250] [funambol.transport.http]
[INFO] [D9044872013116F89220CDEEC2AB9D9A] [] [] []
Request URL: http://192.168.0.74:8080/funambol/ds
[2009-07-03 16:35:27,250] [funambol.transport.http]
[INFO] [D9044872013116F89220CDEEC2AB9D9A] [] [] []
Requested sessionId: null
[2009-07-03 16:35:33,189] [funambol.email] [ERROR] [D9044
872013116F89220CDEEC2AB9D9A] [fwm-35313801010967P7] [] []
Error Opening Connection Store:
```

```
javax.mail.AuthenticationFailedException: AUTHENTICATE
Unknown user or incorrect password
        at com.sun.mail.imap.IMAPStore.protocolConnect(IM
APStore.java:474)
        at javax.mail.Service.connect(Service.java:275)
        at javax.mail.Service.connect(Service.java:156)
        at javax.mail.Service.connect(Service.java:176)
```

In this case, the error message highlighted above is generated by the mail server and therefore might differ from the example. Also in this case, the user will be notified of the error condition.

Summary

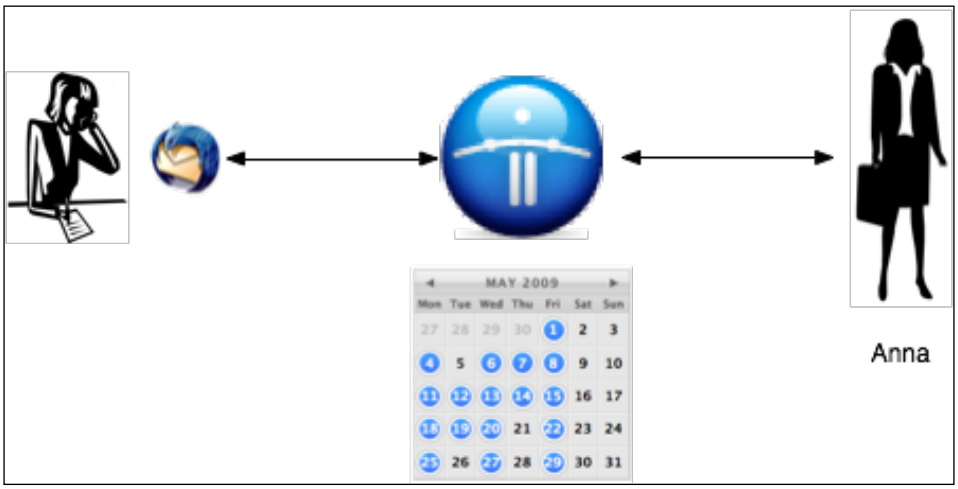
In this chapter, Maria learned how to set up Funambol to deliver the Funambol push e-mail service to her users. This means she configured the E-mail Connector and then created mobile e-mail accounts. Furthermore, Maria acquired familiarity with the Inbox Listener Service, which is the heart of Funambol push e-mail.

Maria also learned how to set up mobile e-mail on a Windows Mobile device and how to access Funambol e-mail.

Lastly, she understood how to improve Funambol e-mail authentication and how to identify authentication issues.

6
Funambol PIM Data Push

Funambol is not only able to push e-mails to a device, but it can also push updates of the address book or calendar. For example, Anna, Maria's manager, would like to receive changes to her calendar every time her assistant changes an appointment. As illustrated in the following screenshot, when Anna is away, her assistant can access her calendar using Thunderbird.

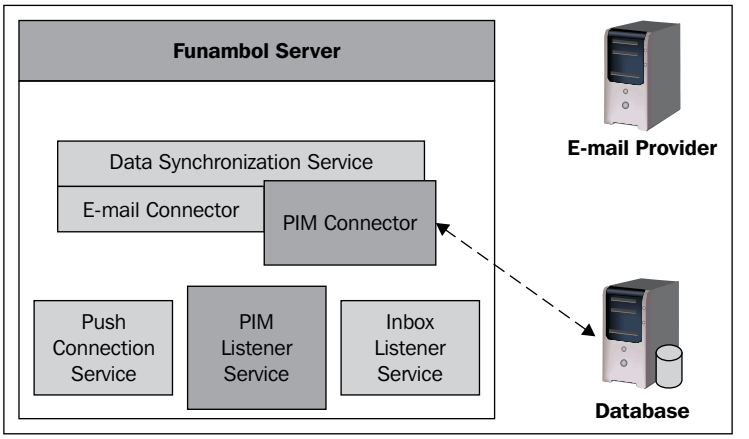


Changes to Anna's appointments are synchronized to the Funambol server and from there the changes are pushed to her mobile phone. The same is true if Anna's assistant makes any changes to Anna's address book.

As for e-mail, this requires a bit of infrastructure setup, which is the next task Maria is going to perform. This chapter explains how Maria can make sure her users get PIM updates on their mobile phones.

The PIM connector

This section focuses on the **PIM Connector**, as shown in the following figure. Before reading through this section, have a look at the figure in Chapter 2 that shows the Funambol architecture.



The PIM Connector has two main components:

- The PIM server extension (represented in the image by the **E-mail Connector** block)
- The **PIM Listener Service**

The former is the extension of the Funambol Data Synchronization Service that allows PIM synchronization via the Funambol database; the latter is the service that detects changes in the user's PIM database and notifies the user's devices.

When the Funambol DS Service receives sync requests for PIM data, the service calls the PIM Connector that updates the server database with the changes sent by the client and sends back to the client the changes made in the database since the last synchronization.

While the server database can be changed from external sources in addition to Funambol itself, changes to the user's PIM data are detected by the PIM Listener Service. When a change is detected, the PIM Listener Service triggers a notification calling the Funambol DS Service that in turn tells the user's device to start a new synchronization.

If, for any reason, the notification cannot be delivered, Funambol remembers that there are changes that have not been synchronized so that as soon as the mobile device becomes available again, the notification will be delivered and the synchronization will take place.

The PIM Listener Service

For PIM push to work, the PIM Listener Service must be up and running.

The default installation of Funambol is already set up to start the PIM Listener Service at startup, when executing the command:

```
$FUNAMBOL_HOME/bin/funambol start
```

However, Maria knows that for her environment, it is recommended that this service be installed on a separate server to support the desired number of users. The easiest way to do this is to install Funambol on a dedicated system and start the PIM Listener Service only.

This is done by executing the following command:

```
> $FUNAMBOL_HOME/bin/pim-listener start
```

As in the case of the Inbox Listener Service, Maria can check that the service is running by issuing the Linux `ps` command and checking that the output is similar to the following:

```
>  ps -ef |grep java
maria      16514     1 36 00:59 pts/1    00:00:03 /
opt/Funambol/tools/jre-1.5.0/jre/bin/java -Dfile.
encoding=UTF-8 -Dfunambol.home=/opt/Funambol -
Dfunambol.pushlistener.config.bean=com/funambol/
pimlistener/PIMListenerConfiguration.xml -Djava.net.
preferIPv4Stack=true -Xmx256M -Dcom.sun.management.
jmxremote -Dcom.sun.management.jmxremote.port=3101 -Dcom.
sun.management.jmxremote.ssl=false -Dcom.sun.management.
jmxremote.authenticate=false com.funambol.pimlistener.
service.PimListener
```

Similar to the Inbox Listener Service, logging in the PIM Listener is configured via the configuration file `log4j-pimlistener.xml` located under the configuration directory `$FUNAMBOL_HOME/config` and not in the Funambol Administration Tool as described in Chapter 3.

For example, to enable full logging, Maria needs to turn the level of the Funambol logger to `ALL`:

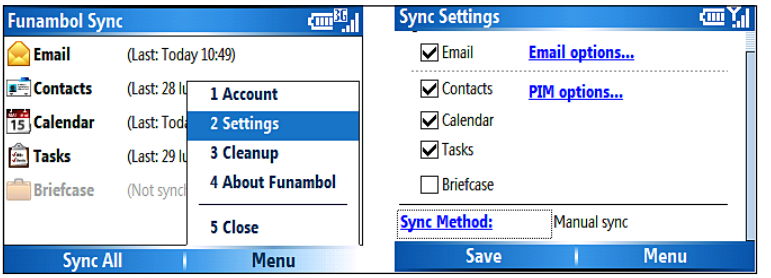
```
<logger name="funambol">
    <level value="ALL"/>
</logger>
```

By default, the PIM Listener log file is located under `$FUNAMBOL_HOME/logs/pim-listener`.

PIM push at work

Maria has seen how to configure a mobile phone for push e-mail. She will now see how PIM push works using the same device.

First, push is totally configurable on the device. Selecting **Settings** from the **Menu**, the following panel appears:

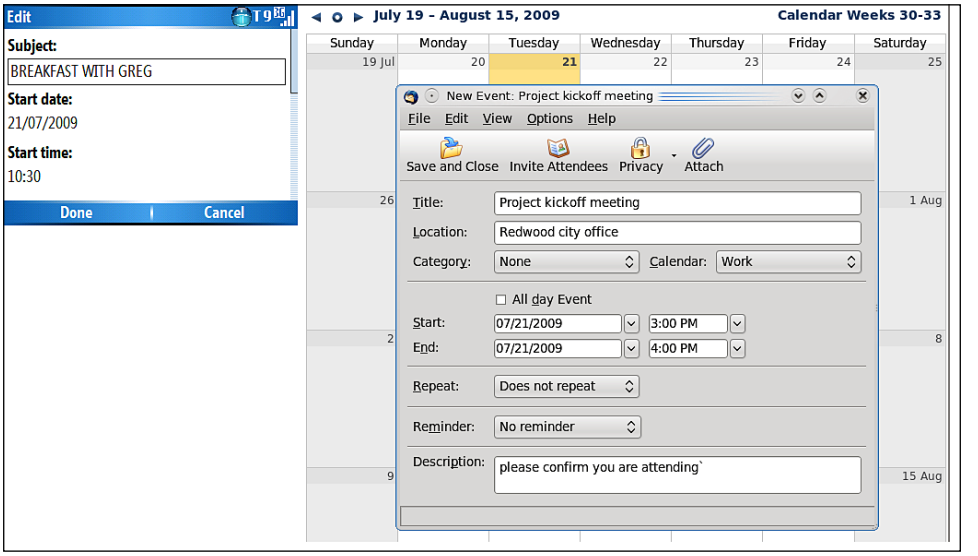


There are three **Sync Method** options available.

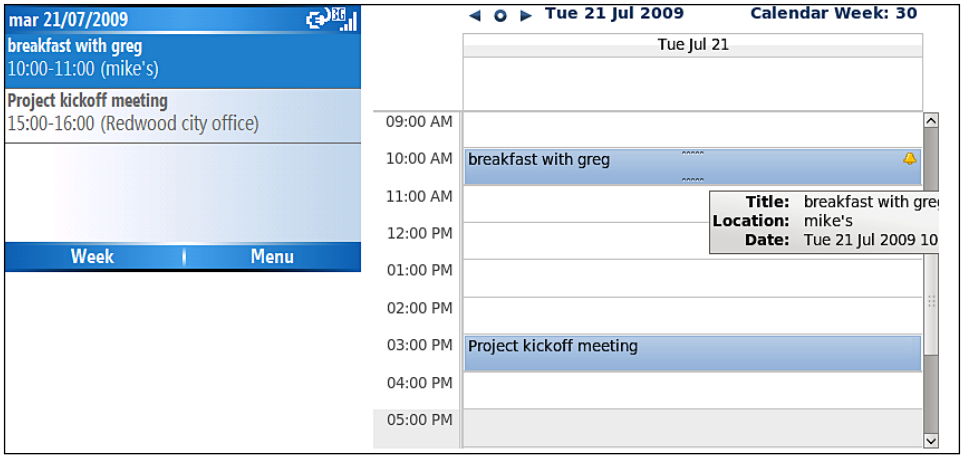
- **Manual sync**: When push is on manual, the synchronization takes place only upon request by the user.
- **Scheduled sync**: With scheduled sync, Maria can select an interval from 15 minutes to once a day. This means that the synchronization will automatically start at periodic intervals based on the selected time lapse.
- **Push service**: For the rest of this chapter, Maria will select the push service so that the synchronization will automatically start whenever updates needing synchronization are present.

Note that the **Sync Method** not only controls server-to-client push, but also client-to-server push. In fact, when the sync method is scheduled or push, all changes on the device are automatically synchronized to the server, even if there aren't any changes on the server.

To see a simple example of PIM push working, Anna adds a new appointment for the morning of July 21 on her device. Her assistant adds a new appointment for the same day in the afternoon in Thunderbird, as shown in the following screenshot:



After a while, synchronization takes place—either because of the change detected on Anna's device or because the PIM listener detected the change made by Anna's assistant on Thunderbird calendar—and both the device and the desktop calendars have the same appointments, as shown in the following screenshot:



Note that to make sure Thunderbird calendar is automatically synchronized, it must be configured with scheduled synchronization on, as described in Chapter 1.

What if connection-less push is impossible

Depending on the mobile environment or carrier infrastructure available for Maria's users as described in Chapter 2, Funambol's connection-less push may be unavailable. This may be due to security restrictions or network configuration, but in some cases the carrier does not provide mobile phones with an IP address that a Funambol server can reach. In these cases, connection-less push becomes impossible and the clients use connection-oriented push instead.

As explained in Chapter 2, connection-oriented push is based on the ability of the clients to establish a constant connection to the Push Connection Service. This is a service that Maria needs to set up and start in her environment. Similarly to other services, it is good practice to have a dedicated system for this service, particularly because the Push Connection Service has a different scalability profile compared to other Funambol services. Given the service it provides, this process is not memory or CPU consuming, but it potentially uses a lot of network resources. This depends on the number of users that needs to be supported, because each device using connection-oriented push will keep a connection open with this system.

The default installation of Funambol is already set up to start the Push Connection Service at startup, when launching the following command:

```
$FUNAMBOL_HOME/bin/funambol start
```

However, in a more scalable environment, it is recommended that this service be installed on a separate server (potentially more than one, depending on the number of users that the system must serve). The easiest way to do this is to install Funambol on the dedicated system and start only the Push Connection Service.

This is done by executing the following command:

```
> $FUNAMBOL_HOME/bin/ctp-server start
```

If the Push Connection Service is not running, users will not be notified of updates using connection-oriented push.

Maria can easily check if the service is up and running by issuing the Linux `ps` command and checking that the output is similar to the following:

```
maria      13194     1  5 10:52 pts/2    00:00:04 /
opt/Funambol/tools/jre-1.5.0/jre/bin/java -Dfile.
encoding=UTF-8 -Dfunambol.home=/opt/Funambol -Djava.
net.preferIPv4Stack=true -Xmx256M -Dcom.sun.management.
jmxremote -Dcom.sun.management.jmxremote.port=7101 -Dcom.
sun.management.jmxremote.ssl=false -Dcom.sun.management.
jmxremote.authenticate=false com.funambol.ctp.server.
CTPServer
```

Push Connection Service configuration

When a client connects to the Push Connection Service, the first thing the service does is authenticate the user. To do this, it connects to the Data Synchronization Service and verifies that the given credentials are correct. Therefore, if the service is running on a different host, the Push Connection Service must be configured with the proper address.

Maria will do this by editing the configuration file `$FUNAMBOL_HOME/com/funambol/ctp/server/CTPServerConfiguration.xml` as shown in the following example:

```
<void property="WSServerInformation">
 <object class="com.funambol.server.admin.ws.client.
               ServerInformation">
  <void property="url">
   <string>http://server:port/funambol/services/admin
   </string>
  </void>
  <void property="username">
   <string>admin</string>
  </void>
  <void property="password">
   <string>password</string>
   </void>
 </object>
</void>
```

Note that after this change, the service must be restarted.

Verifying push activity

To verify that the Push Connection Service is working and pushing users, Maria can review the log file to access a lot of information.

For example, to check that a user device has been pushed, Maria can check for entries like the following:

```
[2009-07-21 15:54:23,209] [funambol.ctp.server.
session-manager] [INFO] [OOB Thread,ctp-notification-
group,127.0.0.1:32876] [/217.202.61.89:1028] [95b54666-
529d-4f56-a0c6-bb1fe5b39a48] [fwm-35313801010967P7]
[anna] Sending notification message to the device 'fwm-
35313801010967P7'
```

If Maria needs to troubleshoot push issues in more detail, she can increase the log level to find a lot more information in the log file, as seen for other services, by editing the file `$FUNAMBOL_HOME/config/log4j-ctpserver.xml` and setting the level for the Funambol logger to `ALL`:

```
<logger name="funambol">
    <level value="ALL"/>
</logger>
```

Summary

In this chapter, Maria learned how to set up and operate PIM push, both from client to server and from server to client. She learned how to set up and start the PIM Connector and PIM Listener Services and how they work in a complex end-to-end scenario using a Windows Mobile phone.

Maria also learned how to run the Push Connection Service, how to verify that it is delivering push notifications, and how to access the log for more troubleshooting details.

7
Synchronizing Devices and Desktops

At this stage, Maria has deployed a complete Funambol system for PIM synchronization and push e-mail, which fits well with the applications and devices that she uses for her job everyday. Specifically, she uses Mozilla Thunderbird as her e-mail, address book, and calendar application, along with a Windows Mobile phone.

However, in her company not all people use Thunderbird or Windows Mobile devices; therefore, Maria faces a more complex and heterogeneous environment. Fortunately, Funambol supports a wide range of mobile phones available in the market, along with one of the most used enterprise desktop applications, Microsoft Outlook.

In this chapter, Maria will learn how to support users with different applications and devices. Let's view the software and phones used by some of Maria's colleagues:

Name	Computer type	E-mail client	Mobile phone
Maria	Linux	Thunderbird	Windows Mobile
Mark	Windows	Outlook	BlackBerry Curve
Andrew	Windows	Outlook	Samsung SGH-Z170
Sonia	Mac OS	Thunderbird	Nokia E61
Brian	Mac OS	Thunderbird	Apple iPhone

In the following sections, we will see how Maria can help get Mark, Andrew, Sonia, and Brian up-and-running with Funambol.

Mark: Outlook and BlackBerry sync clients

Mark uses Microsoft Outlook as his e-mail client and personal data management. His business mobile phone is a BlackBerry Curve. Both Outlook and the BlackBerry do not natively support SyncML, but luckily Funambol provides a Funambol Sync Client for both Microsoft Outlook and BlackBerry. In this section, Mark will see how to use this combination to keep his information in sync.

Funambol Outlook Sync Client

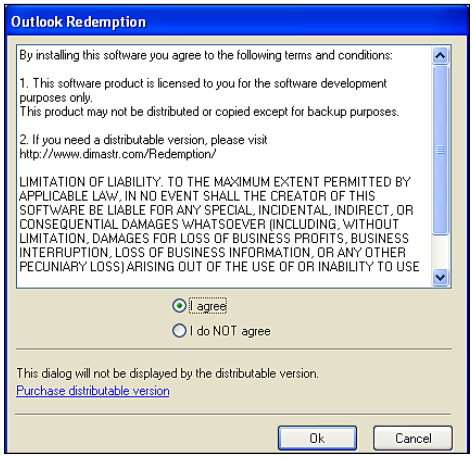
The Funambol Outlook Sync Client can be downloaded from the Funambol download site at `https://www.forge.funambol.org/download`, in the **Desktop Client Software** section, by clicking the following icon:

The browser will ask you to download and save the program `funambol-outlook-sync-client.exe` (with a version number in the file name) and then, depending on the system, it may ask you to run it. On some systems, you may be requested to confirm that you want to run an application downloaded from the Internet. If the system does not start the installation automatically, open the folder where the downloaded file was saved and run it.

Note that Mark will be requested to close Outlook, if it is currently running to perform the installation of the Funambol Outlook Sync Client.

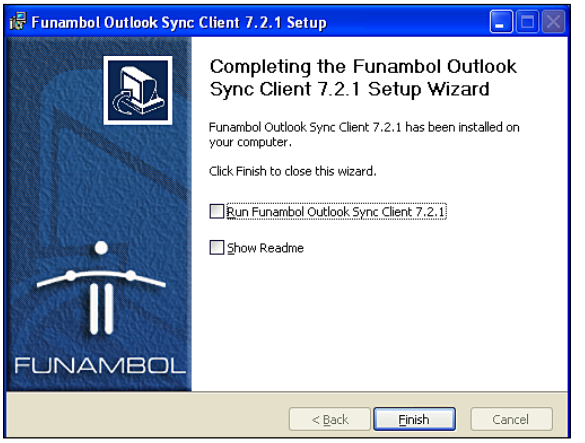
At the present time, the Funambol Outlook Sync Client can be installed on a computer (desktop or laptop) running Microsoft Windows XP or Vista, and may be used with Outlook 2002 (XP), 2003, or 2007.

The installation is straightforward. Mark can simply accept the license terms and other default values. One final step of the setup that deserves to be mentioned is that Mark will be requested to accept the license terms of a library used by the Funambol Outlook Sync Client called **Outlook Redemption** (see `http://www.dimastr.com/redemption` for more information) as illustrated in the following screenshot:

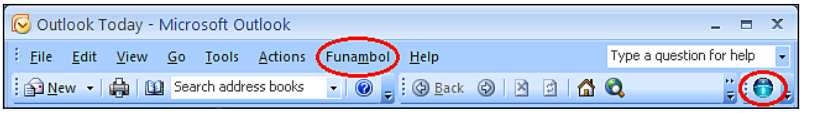


This is used by Funambol to better integrate into Outlook, and is a third-party software not owned or developed by Funambol. For more information about the licensing policy of the Redemption library, see `http://www.dimastr.com/redemption/download.htm`.

The installation of the Funambol Outlook Sync Client ends with the following screenshot:

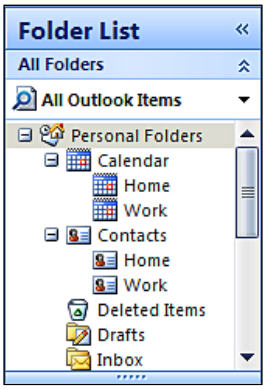


After completing the installation, Mark deselects the option to run Funambol now, as he first wants to see what happened in Outlook. Once the Funambol Sync Client is installed, on opening the Outlook menu, the button bars display the additional items, which are highlighted in the following screenshot:

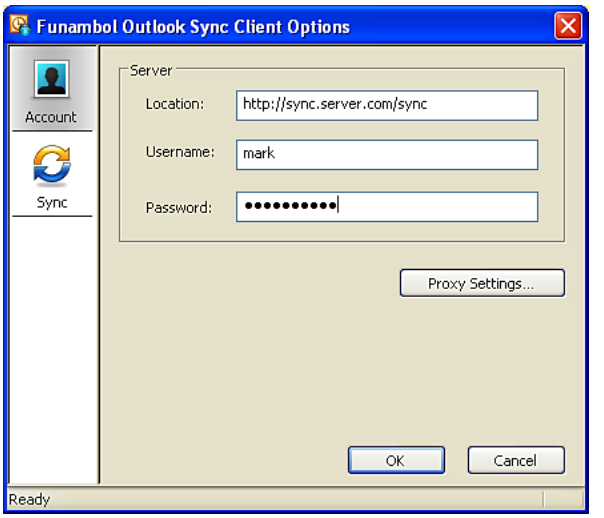


Mark has many contacts and appointments in Outlook. He keeps both his personal and business data in the same application, but he uses the BlackBerry device as his business mobile phone on which he would like to have business contacts and appointments only. This is easily achievable with the use of Funambol and Outlook folders.

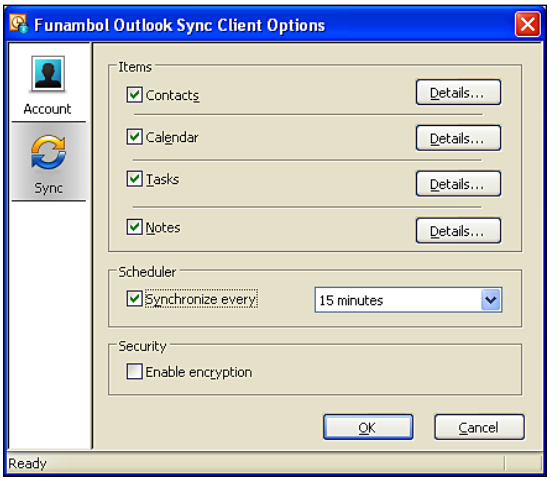
To do this, Mark can create `Contacts` and `Calendar` subfolders, called `Work` and `Home`, as shown in the following screenshot:



He can then put business contacts and appointments in the proper folder based on their purpose and tell Funambol to use the `Work` subfolders instead of the top level ones. The quickest way to open the Funambol client configuration is through the **Funambol** | **Options** menu item that displays the configuration windows shown in the following screenshot. It is possible to switch between the **Account** and **Sync** configuration panels by clicking the icons on the left side.



The following screenshot shows the settings when the user clicks on the **Sync** button.



Account options are the same as in the Funambol Mozilla Sync Client seen in Chapter 1. Mark needs to specify only the **Location**, and the **Username** and **Password** provided by Maria.

Mark can change the folder that the Funambol Outlook Sync Client will synchronize in the Sync options panel by clicking on **Change...** and selecting the `Work` folder from the Outlook **Select Folder** window that pops up (see the following screenshot for **Contacts**).

The same must be done for Calendar. In addition, in the **Calendar** options panel, it is possible to limit the time window for which appointments are synchronized. The possible values are:

- **Future only**
- **Since last week**
- **Since two weeks**
- **Since one month**
- **Since 3 months**
- **Since 6 months**
- **All**

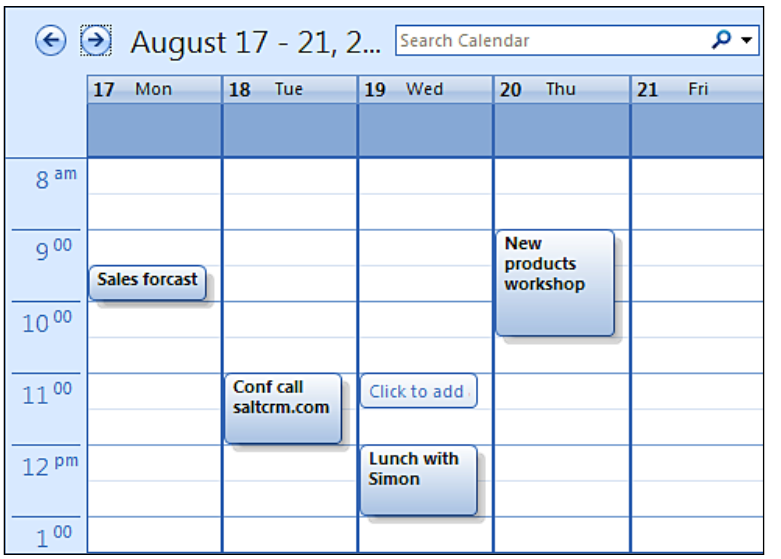
If Mark wants to synchronize only future appointments, **Future only** is the option to select. If he wants to have all appointments in sync, the option to select is **All**. The other options are used to synchronize a specific time window, starting from the specified amount of time in the past to and running to a specified time, which may be in the future. All appointments outside this period will be removed from the phone, but not from the Outlook calendar, which is where Mark wants all of his appointments to be permanently stored.

Another option available in the **Details** section of the **Funambol Outlook Sync Client** options panel is the **Sync Direction**. The most common value for this option is to synchronize the changes in both directions, that is, from Outlook to the Funambol Server and from the Funambol Server to Outlook. However, Mark may decide to synchronize only one-way from Outlook to the server or from the server to Outlook.

In addition to the address book and calendar applications, Mark can also synchronize notes and tasks stored in Outlook.

To see how the Outlook client works, let's assume Mark has the following contacts in his `Work` folder:

and the following appointments in his Work calendar:



Clicking the **Funambol** button in the Outlook button bar, or selecting the **Sync All item** from the **Funambol** menu in the Outlook menu, triggers the synchronization of the selected datasources. Assuming this is the first time Mark has synchronized, the contacts and calendar items above are transferred to the server. Mark is now ready to set up his mobile phone and then leave the office for an important meeting with a customer.

Funambol BlackBerry clients

The first step in setting up Mark's BlackBerry is to download the Funambol software to his device. From the Funambol download page mentioned earlier, Mark clicks on the BlackBerry Email and Sync images in the Mobile Client Software section. Mark is taken to a second download page with different options, as illustrated in the following screenshot:

There are two different applications—the Funambol **BlackBerry Email** Client and the Funambol **BlackBerry Sync** Client. The former is a full push e-mail application, with a look and feel that is consistent with the Funambol suite of mobile clients and that uses Funambol technology for push e-mail. The latter is a BlackBerry extension that makes it possible to synchronize the BlackBerry's native address book and calendar (plus tasks and notes) with Funambol. With both the Sync Clients for Windows Mobile and BlackBerry, Mark can use the native address book and calendar applications on the device, while the underlying data is kept in sync with the Funambol server.

As shown in the previous screenshot, there are two versions of the BlackBerry Sync application—one compatible with BlackBerry OS before version 4.7 and one for more recent versions. As Mark owns a BlackBerry Curve that can be upgraded to the BlackBerry OS version 4.5, he downloads the first version.

To check which version of the OS his BlackBerry device is running, Mark goes to the **Main Menu**, selects **Option** (note that on some devices this is in a folder called **Settings**), and then **About**. The OS version is displayed at the top of this **About** screen.

As Maria wants to provide all her users with a similar experience and wants to be able to support everybody to the best of her ability, the solution that makes more sense for her is to standardize the company on a single sync solution, which is easier than supporting multiple synchronization platforms. At the same time, Mark is happy to know that he can easily switch to another phone and keep using the same data and applications that he already uses. Therefore, Mark downloads and installs both Funambol applications so that they are now available from his BlackBerry main screen using the icons with the Funambol logo as shown in the next image.

To install the clients, Mark needs the Blackberry Desktop Manager 4.5, which he has installed on his PC from the BlackBerry software CD included with the device, and with some support from Maria. In fact, the clients downloaded from the Funambol download page are set up to be provisioned over the air or via the Internet. This would require Maria to set up a website from which the clients can be downloaded on the device and automatically installed by the phone. However, Maria prefers to manually install the two clients on Mark's device from Mark's PC to simplify the process.

Before installation, Maria unzips the files downloaded earlier into a directory of her choice—for example, `c:\blackberry\email` for the e-mail client and `c:\blackberry\sync` for the sync client. Maria opens a command line window and then issues the following commands:

- To install the Funambol Email Client

```
cd c:\blackberry\email\BlackBerry\os42\native
JavaLoader -u load *.cod
```

- To install the Funambol BlackBerry Sync Client

```
cd c:\blackberry\sync
JavaLoader -u load *.cod
```

JavaLoader is an installation program available with BlackBerry Desktop Manager from versions 4.5 and above.

Once installed, the two new Funambol applications are available from the BlackBerry's main screen as shown next:

Funambol BlackBerry Sync Client

Like other Funambol clients, the BlackBerry Sync Client is straightforward to configure. All Mark needs to do is launch the application from the main screen and open the configuration panel to insert the account settings.

Once the client has been configured, selecting **Sync All** from the menu downloads the contacts and appointments previously synchronized with Outlook. The result will be as shown in the following screenshot:

Mark is now in sync and can leave his desk confidently, as his business data will always be with him. The last step for him is to set up and run the e-mail client.

Note that Mark's BlackBerry device does not have separate address book or calendar folders. All items on the phone will therefore be synchronized with the specified Outlook folders.

Funambol BlackBerry E-mail Client

Going into the details of the Funambol BlackBerry E-mail Client is beyond the scope of this book. This application is a basic but powerful e-mail client that uses SyncML as the protocol to communicate with the Funambol server, which in turn uses IMAP or POP to talk to the e-mail server.

For more information on how to use the BlackBerry E-mail client, Mark can refer to the user guide, which he can download from the documentation download page at `https://www.forge.funambol.org/download/documentation.html`.

As usual, the first thing Mark is asked to do upon starting the application is specify the account settings. After everything is set up, Mark starts to receive his e-mail on the phone, as shown in the following image:

Andrew: Outlook and a Java phone

Andrew's case is another very common scenario. Like Mark, his e-mail client and personal information management system is Microsoft Outlook, but instead of a BlackBerry he owns a Java phone, the Samsung SGH-Z170. This phone has a powerful Java platform, but does not have a native SyncML client preinstalled. Nevertheless, Funambol can still be of great help to Andrew, thanks to the Funambol Java ME Email Client. This is an e-mail client similar to the one for BlackBerry, but it is mainly designed for Java phones.

Due to the variety of Java phones available, Maria needs to do some preparation work to enable Andrew's phone for Funambol push e-mail. As usual, the first step for her is to download the Funambol Java ME Email Client from the Funambol download page (again, in the the Mobile Client Software section). This package includes a pool of different versions of the client for different mobile phones. To know which version to use for the Samsung SGH-Z170, Maria opens the `email_client_readme.txt` file in the root directory of the archive. The different versions are manufacturer specific, but sometimes there could be more than one version for a single phone model. Maria might need to further refer to the manufacturer's website or to the device's user manual together with the aforementioned readme file to select the correct package.

Once Maria has identified the right version for Andrew's phone, she transfers it to the mobile phone and starts the installation process. Again, she may need to refer to the phone's user manual to know how to install third-party applications on the phone. In the case of Andrew's phone, Maria needs to install the package named `SamsungHighRes_T` (the "_T" labeled package contains the signed version of the application).

Upon the first execution of the client, Andrew will be asked to enter the account information—server URL, username, and password. After Andrew's credentials are verified, his e-mail will be downloaded on the phone.

Andrew's inbox will look like the last figure in the previous section. Andrew can also further configure the Funambol Email client like in the case of the BlackBerry client.

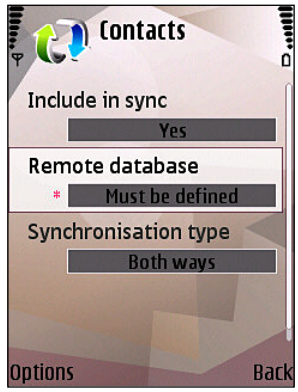
Sonia: MacOS and a SyncML phone

Sonia is a Mac user; she will never opt for a different system as she has been a Mac addict for a long time. Sonia is part of the technical support team; she has a technical background and loves open source software. Her e-mail and personal data client is Mozilla Thunderbird, so she can use Funambol software for PIM synchronization. Sonia's mobile phone is a Nokia E61, a perfect device for her, who needs to be in touch with her team even when she is not in the office, and sometimes even outside normal office hours.

The good news for both Sonia and Maria is that there is a version of the Funambol Mozilla Sync Client for Mac OS too. Sonia can download it from the Mozilla add-on website as described in Chapter 1. Doing it from her laptop, the site will present a download page with the Funambol client version that matches her system.

Sonia's phone, a Nokia E61, has a SyncML client installed out of the box; therefore no additional software is required to keep her personal data in sync. To configure the phone's SyncML client, Sonia needs to create a new SyncML account from the Data Synchronization application, placed under **Menu | Tools | Sync**, which will result in the following screen: Sonia creates a new account selecting **Options | New Sync Profile** specifying the connection information.

On phones such as the Nokia E61 (Symbian series 60 third Edition), Sonia needs to specify a piece of information that is not usually necessary for Funambol clients—the "remote" name of the sources or databases to synchronize. These values are used to address the server databases that must be synchronized with the mobile phone, as different servers may use different names. These phones can synchronize data for different applications: Contacts, Calendar, Notes, Lifeblog, Text messages, and Bookmarks. Funambol natively supports Contacts, Calendar, and Notes and hence Sonia needs to specify the remote names for such applications. For example, in the case of Contacts, selecting **Applications | Contacts** from the profile settings panel, will bring up the following screen:



In the case of a standard installation of a Funambol server, these names are "card" for contacts, "cal" for appointments and tasks when stored in the same database on the phone (the phone does not provide the possibility to specify a separate name for task synchronization), "event" for appointments when tasks are synchronized separately, "task" for tasks and to-dos, and "note" for notes synchronization.

For her Nokia E61, Sonia uses the following names—"card" for contacts, "cal" for appointments and tasks, and "note" for notes synchronization.

Starting the synchronization process is fairly simple; Sonia simply needs to open the application and select **Synchronize** from the **Options** menu.

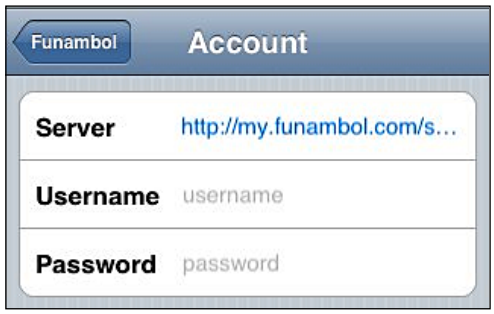
E-mail on the Nokia E61

The Nokia E61 is a powerful smartphone that provides e-mail client. However, this application has a couple of drawbacks that make it difficult to use. Firstly, the client is not very user-friendly. Secondly, it requires Maria to open POP or IMAP in her company's network, something that Maria is not willing to do for security reasons. Funambol is the solution to these issues and the Nokia E61 is a great Java phone. Therefore, all Maria and Sonia need to do is install the Funambol Java ME Email Client. The package to be downloaded and installed in this case is `NokiaE61_T`.

Brian: MacOS and iPhone

One of the most successful phones on the market is the Apple iPhone. Brian chose Apple for both, his laptop and his mobile device. On his Mac, being a Mozilla Thunderbird user, he uses the Funambol Mozilla Sync Client with the credentials that Maria provided him.

Thanks to Funambol software, Brian can also bring his Thunderbird address book with him on the iPhone. This is done by the Funambol iPhone Sync Client, which can be downloaded from the Apple Store at this location: `http://itunes.com/app/funambol`. Once it is downloaded and installed, on first launch, the client will request Brian's credentials and the server's URL. It is then ready to synchronize, as shown in the following screenshot:



To synchronize the iPhone's address book with the server when necessary, Brian simply needs to start the application and press the icon in the center of the application's screen, as shown in the following screenshot:

At the end of the synchronization process, Brian's mobile address book and Thunderbird address book will contain the same contacts.

Apple does not currently provide programmatic access to the native iPhone calendar application, so there is no Funambol calendar sync application yet. Some features such as scheduled synchronization which are available in other Funambol clients, are not available on the iPhone due to the limitations of the iPhone platform.

Summary

In this section, Maria, Mark, Andrew, Sonia, and Brian learned how to use Funambol to keep their data in sync between their PC, server, and mobile phones. They saw different combinations of systems and devices, which reflect many enterprise environments where different users work with different systems and applications. This is particularly true in the mobile space, as those devices are often personal equipment that people choose based not only on business needs but also personal preferences.

Maria's users learned how to configure their systems and Funambol clients to set up synchronization and push e-mail. Maria is now under control of a full synchronization and mobile e-mail solution for her company, all in one system that she can manage and maintain.

8
Making the Most of Funambol: Community and its Projects

In the previous chapter, Maria and her team learned how to synchronize their data and receive e-mails on different devices. They now understand the advantages of a multiplatform solution that allows everyone to keep using the systems and tools they are familiar with, maintaining a similar user experience. In this chapter, Maria will learn that Funambol has another incomparable advantage—it is an open source software solution. Maria can take advantage of the vast Funambol community to satisfy various needs, from the availability of technical support to the wide range of projects that the community makes available. In addition, to learn more about the Funambol community, Maria will be guided through an excellent example which will help her to understand the power of the community. She will learn how to integrate Funambol with SugarCRM, the most popular open source **customer relationship management** (**CRM**) system.

Integrating Funambol and SugarCRM

SugarCRM (`http://www.sugarcrm.com/crm`) is a popular open source CRM system for enterprises. Maria has deployed this CRM system for the company's sales department. Therefore, the next step is to make sure that sales people take their contacts and calendars with them when they are on the road.

Installation and configuration

It is beyond the scope of this book to describe SugarCRM in detail or explain how to set it up. For the rest of this chapter, we will assume that Maria has a good understanding of SugarCRM in her business environment. However, to ease the understanding of the integration, Maria is going to start a clean installation of SugarCRM in a staging environment and she will experiment with the integration using the default dataset created at installation time.

SugarCRM Community Edition installation

First, Maria downloads the latest version (5.02j at the time of writing) of SugarCRM Community Edition from `http://www.sugarcrm.com/crm/download/sugar-suite.html`. She previously set up the machine with the following required software:

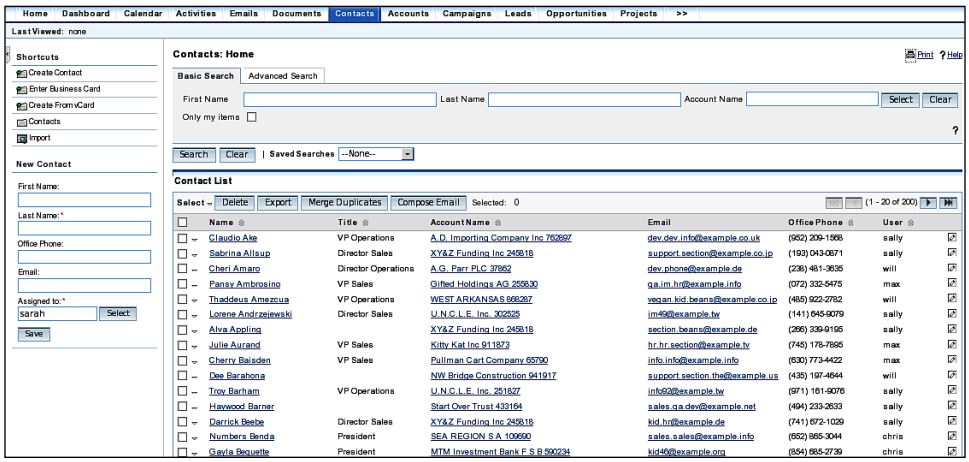
- Web server with PHP support
- MySQL database management system with a proper user and database
- PHP support for MySQL

The program is downloaded in the form of a ZIP, extracted to a directory, which will be served by the web server. This directory is the one used to access the application, so Maria chooses to name it `sugarcrm`. Maria then completes the installation with the setup wizard available at the URL `http://localhost/sugarcrm/install.php`.

During setup, Maria selects the following options:

- MySQL database type
- Database name and connectivity information (as per the MySQL setup)
- Created demo data
- SugarCRM admin password
- Local settings

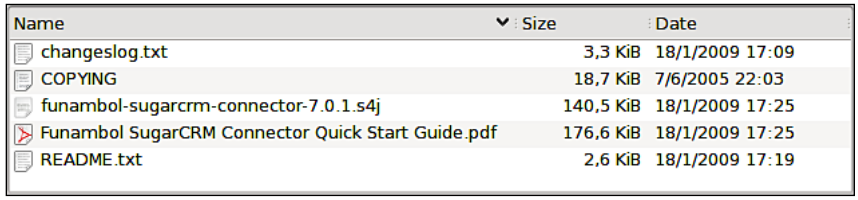
After completing the installation, thanks to the creation of the demo dataset, SugarCRM already contains data which can be used to test the integration. For example, the following screenshot shows the address book which appears when someone logs in with the name Sarah:



SugarCRM Funambol Connector installation and configuration

Now that Maria has set up the staging environment, it is time to discover how to connect it with Funambol. The answer is a community project called the **Funambol-SugarCRM Connector**, which is an open source project hosted on the SugarCRM forge website at `http://www.sugarforge.org/projects/sync4j`.

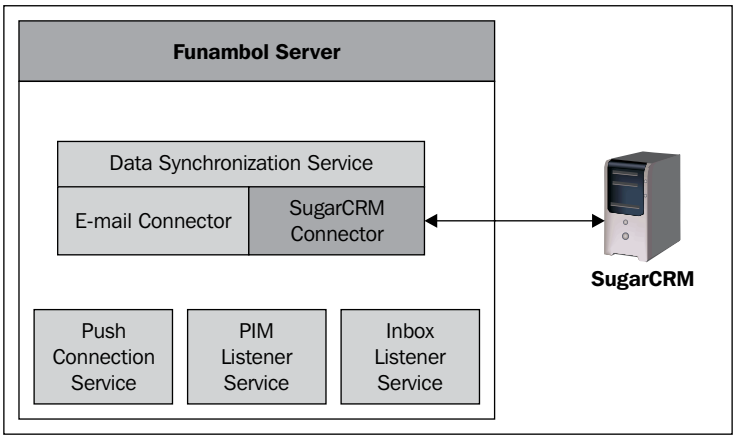
The packaged version of the connector can be downloaded from the **Download** section of the project site. At the time of writing, the latest version is distributed in a ZIP file called `funambol-sugarcrm-connector-7.0.1.zip`, whose content is illustrated in the following file system view:



Maria can use the **Funambol SugarCRM Connector Quick Start Guide** as reference for the installation of the connector, but before starting this task, it is worth learning a bit more about third-party Funambol connectors.

With reference to the architecture illustration of Chapter 2, Funambol uses a connector architecture so that developers can connect the Funambol Server to any kind of backend without knowing anything about devices and synchronization concepts.

Funambol has an out of the box implementation of a PIM connector that accesses PIM data stored in a database. The SugarCRM connector replaces the Funambol PIM Connector with a module that contains the logic to access a remote SugarCRM instance, as illustrated in the following figure:



This can be accomplished because SugarCRM exposes an HTTP-based API that can be used by external software to interoperate with the CRM system.

Maria is now ready to install the connector, following these steps:

1. She expands the archive file into a directory of choice and then copies the connector file, `funambol-sugarcrm-7.0.1.s4j`, to the Funambol modules directory `$FUNAMBOL_HOME/ds-server/modules`.
2. She opens the file `$FUNAMBOL_HOME/ds-server/install.properties` using a text editor. At the very bottom of the file there is a line that starts with `modules-to-install=...;` she adds `funambol-sugarcrm-connector-7.0.1` at the end of the list, making it look like the following example:

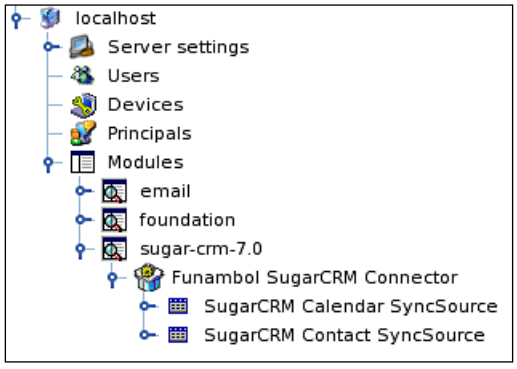
   ```
   modules-to-install=content-provider-7.1.2,
   email-connector-7.1.1,foundation-7.1.1,phones-support-
   7.1.0,webdemo-7.1.0,funambol-sugarcrm-connector-7.0.1.
   ```
3. She launches a command shell and runs the following command:

   ```
   $FUNAMBOL_HOME/bin/install-modules
   ```

The last step launches an installation procedure that requires Maria's input. Basically, all modules are refreshed and Maria is asked if the database for each module must be created. Maria does not need to recreate the database for any of the modules, except the SugarCRM connector. When the installation displays the following output, Maria can press *Y* so that the connector installation will register the module inside Funambol.

```
     [echo] funambol-sugarcrm-connector-7.0.1
installation...
      [echo] Database installation for module funambol-
sugarcrm-connector-7.0.1 on hypersonic (/opt/Funambol.
sugar/ds-server)
   [iterate] The Funambol Data Synchronization Server
installation program can now create
   [iterate] the database required by the module funambol-
sugarcrm-connector-7.0.1 (if any is needed).
   [iterate] You can skip this step if you have already a
valid database created
   [iterate] or the module does not require a database.
   [iterate] If you choose 'y' your existing data will be
deleted.
   [iterate] Do you want to recreate the database?
  [iterate]                (y,n)
```

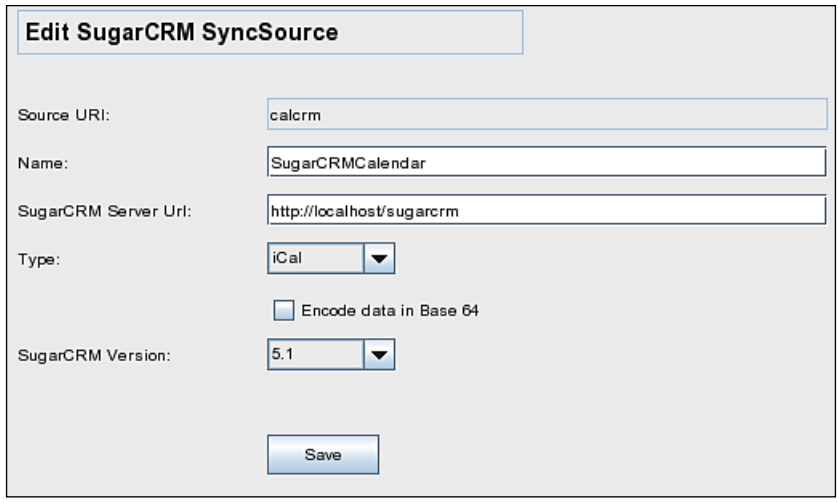
The SugarCRM Connector is now installed. When the server is running, it appears in the connector list in the Funambol Administration Tool:



The final step for Maria is to use the Funambol Administration Tool to configure the new SugarCRM SyncSources to connect to the SugarCRM server.

A SyncSource represents a particular datasource provided by a connector. For example, the SugarCRM Connector provides connectivity to SugarCRM contacts and calendar that are linked to the CRM address book and calendar applications.

Maria now needs to create the SyncSource to access the staging installation of SugarCRM. To do so, Maria double-clicks the **SugarCRM Calendar SyncSource**, opening the following form and inserting the information as shown in the screenshot:



Note the **Source URI** value. This is the identifier of the datasource that will need to be set in the configuration of the client.

Similarly, Maria creates a SugarCRM Contact SyncSource with the following information:

- **Source URI**: **cardcrm**
- **Name**: **AddressBookSugarCRM**
- **SugarCRM Server Url: http://localhost/sugarcrm**
- **Type**: **vCard**
- **SugarCRM Version**: **5.1**

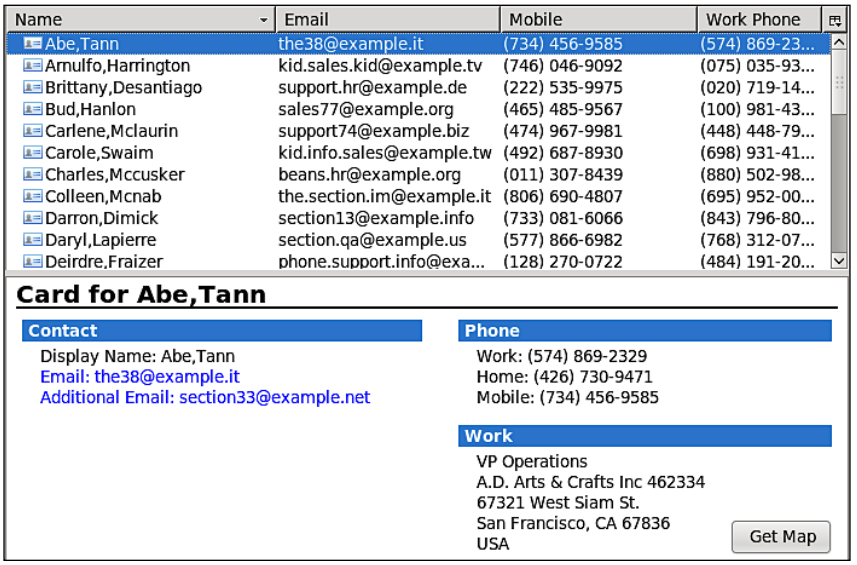
Finally, Maria must tell the Funambol server to authenticate the users on SugarCRM. To do so, she double-clicks the server settings tree, opening the **Server settings panel** and sets the **Officer** field to `com/funambol/server/security/SugarcrmOfficer.xml`.

PIM synchronization

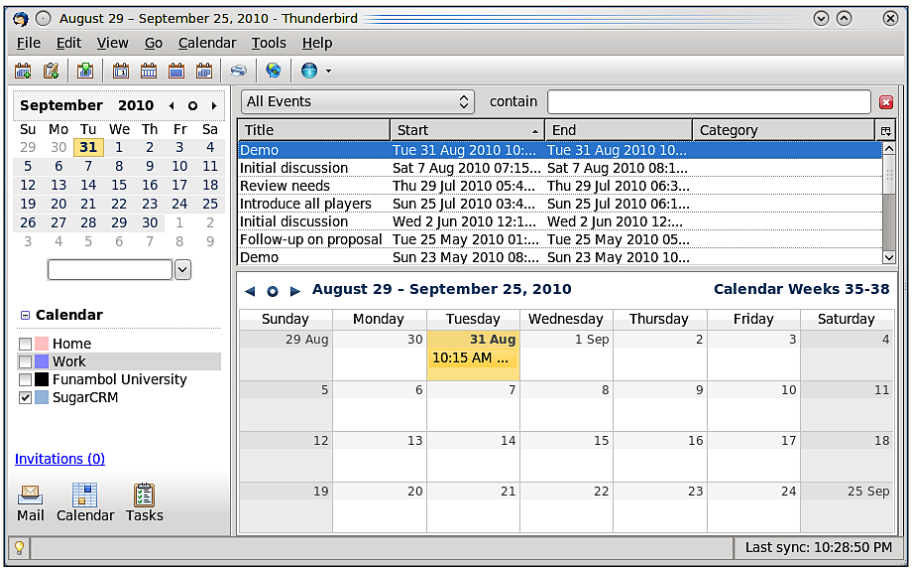
Now, Maria will test the synchronization of Thunderbird's address book and calendar with SugarCRM. First, she configures the Funambol Mozilla Sync Client with the following information:

- Account
    - **Location: http://<stagingserver>:8080/funambol/ds**
    - **Username: sarah**
    - **Password: sarah**
- Sync/Contacts/Details
    - **Remote name: cardcrm**
- Sync/Calendar/Details
    - **Calendar remote name: calcrm**

Maria then starts the synchronization. Once the sync has completed, her address book will appear as follows:

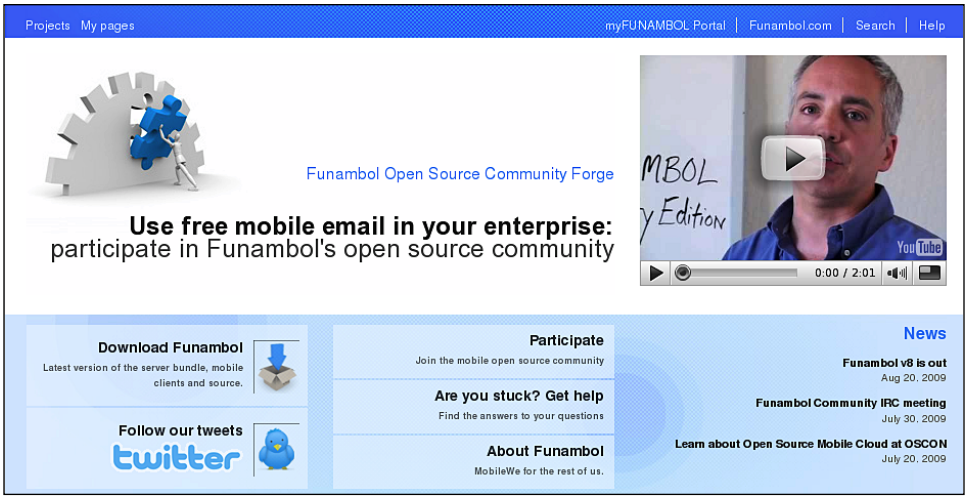


And the calendar will appear as shown next:



The Funambol community

One of the most important aspects of a community is a place to meet. In the days of the Internet, this place is a website—the Funambol Forge (`http://forge.funambol.org`).



This website is the starting place for people to follow Funambol news and events, get user and development help, participate in various discussions, and take part in the development of the software. This is the beauty of open source—users, developers, or people who are just curious, can get together to talk about the software and share their ideas and experiences. Users help each other, and together they greatly contribute to making the software better for their needs.

The Funambol community can be divided into end users, developers, and core developers. **End users** are people who use the services provided by the software and also operations staff responsible for the deployment and maintenance of the service. **Developers** are those who build Funambol extensions on top of Funambol SDKs and APIs. These may range from custom integration development to open source projects such as connectors and clients. Finally, **core developers** are the developers of the Funambol software and its core components.

Funambol Forge

The Funambol Forge is the common place where Funambol users can chat and share information, from general discussions to code. It is not just a forum, it also provides a full range of tools that developers can use to start their own development projects on the Funambol platform. These range from the download of the software, to development tools, and new open source projects. These powerful tools will be introduced in the following sections.

Software download

This is a section of the website that Maria is already familiar with. As seen in the previous sections, Maria has used this service to download all the necessary software. It contains all the latest packages of Funambol, plus additional information on community projects and additional packages such as the source packages.

Each download can be completed anonymously, although it is appreciated when people provide optional background information in the download form. This information helps Funambol improve the software.

It is worth noting that from the download section of the Forge, only the latest version of Funambol can be downloaded. If, for any reason, an earlier Funambol version is needed, it can be downloaded from the hosting facility at: `http://forge.ow2.org/project/showfiles.php?group_id=96`.

Getting help

Open source Funambol software relies on the dedication of developers and the experience of users. Sharing of knowledge is essential, as are user comments and contributions. Funambol's success can be greatly attributed to the global community of Funambol users that support each other.

At the same time, Maria has access to a great source of help and an opportunity to contribute to the growth of the community in a positive feedback loop that results in higher quality software.

In this area of the Forge (`https://help.forge.funambol.org`), Maria can find online documentation, information on each project (Funambol or community), and links to the discussion forums.

There are two main discussion forums for the core Funambol software:

- **Open discussions forum**: This forum includes topics started by Funambol users and general developers
- **Developer's discussion forum**: This forum includes topics that interest Funambol core developers

Once Maria has created a forge account, she can access these forums using a web interface or e-mail, as an old-fashioned e-mail list.

Other important sources of information are the **Top Answers** and the **Frequently Asked Questions**, accessible from this section of the Forge website.

Participate

This section of the Forge explains how to participate in the development of Funambol. People can participate in different ways. Contributing code is the most common way of participating, but non-developers can contribute with testing, helping other people, translating the software in different languages, and so on.

Funambol, and its community members, agree to honor the Funambol Open Source Project Social Contract, which is reported in the next list (the latest version can be found at `https://www.forge.funambol.org/participate/socialcontract.html`):

As a member of the community, Funambol:

Contributes code and documentation to the project.
Provides the forge and generates the binaries:

- Integrates community code and documentation contributions into each release
- Makes source code available via popular and appropriate open source licenses
- Maintains back-revs of the source and binaries for two years after release

Provides vehicles for community interaction such as discussion services:

- A Core Developers mailing list for all contributors to Funambol code and platform extensions.
- An Open Discussion list to discuss future development, SyncML issues and everything useful for Funambol Community.
- Help Project a knowledge base and forum for installation problems, configuration issues and user questions.

Resolves bugs when they are identified.
Integrates feature requests.
Supports the community and responds promptly to requests.
Helps grow, inform, and unify the community:

- Communicates regularly with the community
- Organizes and implements developer and community programs
- Recognizes top contributors
- Sponsors related groups (if you want to have a local user group, talk to us about sponsoring it)

Users and members of the open source community

You may use the software as you see fit, following Funambol open source licensing norms.

You may also develop, extend, and create bug fixes following open source license norms. Read about licensing to understand more. Also, Funambol pays for extensions of the code base via the Funambol Code Sniper Program.

It is requested that you provide your e-mail address when you download, though it's completely optional.

- "Beta" releases are the most bleeding-edge versions of the code. They have bugs but they also have the latest feature and performance enhancements.
- The "Stable" release is just that, stable. It has fixes for bugs found by the community.

Many people choose to post questions and answers to the mailing lists. For developer-oriented issues, participate in the developer mailing list. If you've discovered a work-around or fixed a bug, please post it.

- Code or documentation improvements, as well as project extensions, are covered under our open source licensing terms (`http://funambol.com/solutions/licensing.php`).
- Read How to Contribute to the Funambol Open Source Project (`https://core.forge.funambol.org/servlets/ProjectProcess?pageID=1637&subpageID=9YmJ6H`) to learn more.

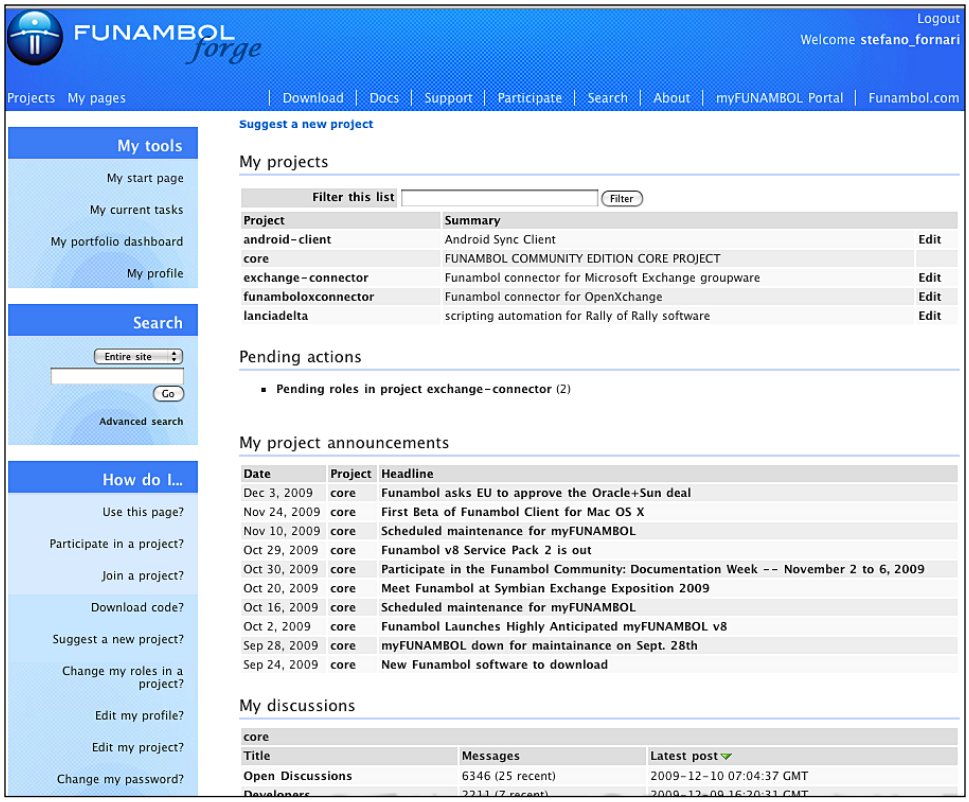
Another important way to participate in the Funambol community is by submitting bugs. This is done using a tool called bug tracker which, for historical reasons, is not part of the Forge. Therefore, if Maria finds a bug, she can go to `http://forge.ow2.org/tracker/?atid=100096&group_id=96&func=browse`, search if the issue has already been reported and, if not, file a new issue.

News and information

There are many ways to stay in touch with Funambol. The Forge homepage summarizes the most recent news from different sources. Another channel is Twitter (`http://twitter.com/funambol`). Also, when downloading Funambol software, the user can sign up for the Funambol e-newsletter. The discussion boards are also a powerful source of information and news about the software and community.

Personal workspace

It is a convenient way for Maria to create a personalized workspace so that she can have all she is interested in about Funambol in one place. This can be done after creating an account on the Forge and then clicking on the **My pages** link. This brings her to a customizable page, as illustrated in the following screenshot, where she can follow her projects, news, and discussions:



Projects

One of the most important features of the Forge is that it is a powerful tool to host community projects. A community project can be started and maintained by potentially anyone in the Funambol community. There is no direct intervention by Funambol personnel or core developers. Funambol simply provides the tools to facilitate the community to develop the project. This means providing a way to store and share the code through an online Subversion repository, communication through discussion forums, and documentation through a dedicated wiki. Each community project has its own space and set of tools.

If Maria wants to start a community project to develop a connector to a backend that is not natively supported by Funambol, all she needs to do is create a Forge account and suggest a new project from her personal page.

At the time of writing, the Funambol Forge hosts the following projects:

Name	Description
akunambol	Funambol client for Akonadi
android-client	Android Sync Client
atollon Funambol	Connector to sync Atollon groupware
avatargrabber	Grabs the avatar of your contacts from social networks
core	FUNAMBOL COMMUNITY EDITION CORE PROJECT
db-connector	Enables synchronization for DB tables
dc-connecter	A customized connecter for synching contacts with dc server
dm	Funambol device management server
domino-connector	Funambol Connector for Domino server
exchange-connector	Funambol connector for Microsoft Exchange groupware
facebook-client	Plugin to sync facebook contacts and events

Name	Description
fcppunit	Funambol tiny CPP Unit
google-connector	Funambol Google plug-in and connector
groupdav-connector	Funambol GroupDAV Connector
jajah-connector	Funambol Jajah PIM plug-in
jsr75-plugin	This module will import both SIM address book and phone
jsr75toolkit	A J2ME test program for JSR75 API capabilities
knowledgetree-plugin	Integration of Funambol with KnowledgeTree
l10n	L10N Sniper program
lanciadelta	Scripting automation for Rally of Rally software
ldap-connector	Funambol connector for LDAP
lotus-client	Lotus Organizer Connector
mac-plugin	Funambol plugin for Apple Mac desktop applications
mozilla-plugin	Funambol Mozilla Sync Client
openlemonade	A project to investigate the OpenLemonade protocol
palm-client	Sync Client for Palm OS
phonesniper	Test phones
pysyncml	Python binding for Funambol C++ API
rtm-sync-client	Synchronization client for "Remember The Milk" service
samsung-instinct	Enable e-mail
scalix-connector	Funambol connector for Scalix
sogo-connector	Funambol connector for SOGo
sts	Source-to-Source Synchronization Service
tinyerpconnector	A connector to synchronize Tinyerp contacts
ws-connector	Funambol connector for web services
yahoo-connector	Funambol Yahoo plug-in and connector

Some projects have subprojects. A project of this type is a container of other projects, each with its own space. The most important collection of projects is "core", which contains the majority of all Funambol core components. The subprojects of the core project are:

Name	Description
apple-clients	All the funambol Sync clients for Apple's platforms
blackberry-client	Funambol BlackBerry Sync Client
client-sdk	Funambol Client SDK
funamboloxconnector	Funambol connector for OpenXchange
ipod-client	Funambol iPod Sync Client
jme-client	Funambol JavaME Email Client
outlook-client	Outlook Sync Client
sdk	Funambol SDK
symbian-client	Funambol Symbian Sync Client
windowsmobile-client	Windows Mobile Sync Client

Funambol community programs

Funambol strongly believes in the power of the community and particular in the mobile space, considering the diversity of the available devices and how they are used in different business environments.

To encourage people to make valuable contributions to the entire community, Funambol started three Community Programs called snipers: Code Sniper, Lion Sniper, and Phone Sniper.

Code Sniper

Funambol maintains a wish list of additional software components that community members would like to have. To encourage community members to work on these components, Funambol offers bounties to the developers who deliver certain projects. It's a great way to learn more about mobile software and to help other community members, while being economically rewarded for their work.

One of the goals of the the program is to broaden interoperability of Funambol to connect with as many useful, widely-requested applications and services as possible (based on feedback from the Funambol Community). This is not only interesting for developers, but also for Maria because she knows that she may find what she needs as result of this program (as seen earlier with SugarCRM). Another possibility for Maria is to propose to add new sniper projects.

Another important goal of the Code Sniper is to recognize community members who developed connectors, sync clients, and so on, with the aim of encouraging the start of new projects that the community can take over, improve, and maintain.

Finally, the Code Sniper is a great way to influence the direction and reach of the Funambol project through direct contribution of open source code.

More details about the Code Sniper program can be found here: `https://codesniper.forge.funambol.org`.

Lion Sniper

Lion Sniper is a community program that is intended to make Funambol's mobile sync and push e-mail more accessible for non-English speakers. The name *Lion* Sniper is a play on the term *L10n*, which is a technical nickname for *localization* that uses the number 10 between the letters *l* and *n*. The Funambol Lion Sniper program provides a bounty to community members who want to localize Funambol Mobile Sync and Push E-mail open source clients into other languages.

Maria works for a multinational company, so she may need clients in different languages. Some of these clients may already have been supported, but if this is not the case, she has the opportunity to make the translation internally and contribute it to the community for the benefits of all users.

The Lion Sniper is designed to:

- Translate and adapt Funambol open source e-mail and sync clients to other languages
- Reward community members that contribute to the global Funambol community
- Enable increased participation from non-English speaking users

More details about the Lion Sniper program can be found here: `https://l10n.forge.funambol.org`.

Phone Sniper

One of Funambol's greatest strengths is its broad support of many mobile devices, including SyncML, Windows Mobile, and Java ME-based handsets. The Phone Sniper program rewards community members by offering to pay them for testing their mobile devices with Funambol to ensure compatibility.

More details about the Phone Sniper program can be found here: `https://phonesniper.forge.funambol.org`.

Funambol license

Funambol uses a dual licensing model that is common with commercial open source companies, consisting of both open source and commercial licensing. For open source licensing, Funambol uses AGPLv3 (`http://www.fsf.org/licensing/licenses/agpl-3.0.html`) for server and connector code, and GPLv3 for client code. If Maria agrees with using this kind of license inside her organization, she can use the software as described in the previous chapters.

If, instead, Maria's company policy is not to use AGPL software, she can acquire a commercial license which provides access to additional features. For details, Maria should refer to the Funambol website and for more on licensing information, she can visit the page at `http://funambol.com/solutions/licensing.php`.

This is just a brief introduction on the Funambol license. It is not meant to contain specific legal information or advice. If Maria has specific legal questions or needs, she should sort them out with her legal department.

Summary

In this chapter, Maria learned how to connect Funambol to her enterprise CRM system using an open source community project. She also learned how the Funambol community can help her and her users, and how she can contribute back to the community. This gives her the advantage of being able to share Funambol knowledge and experience with many other users, administrators, and developers, thus lowering the risks of being stuck waiting for an issue to be resolved and, in fact, increasing the chances of finding exactly what she needs.

This chapter is also the natural ending point of the journey to describe the use of the Funambol mobile open source platform in an enterprise environment. It started from the basic need of bringing user address books and calendars to their mobile phones and ventured all the way to integration with the company's CRM system. All of this is powered by open source and the Funambol community.

Part 2

Extending Funambol

Introduction to SyncML

Extending the Funambol Data Synchronization Service

9
Introduction to SyncML

The final part of this book will provide deeper coverage of some technical aspects of the Funambol platform.

This section is more informative than the first part of the book, and Maria will probably skip it. However, her development team will find this additional information particularly useful to expand the possibilities of the Funambol platform beyond the immediate needs of personal data synchronization and mobile e-mail. The horizon of possible applications that can be mobilized is wide and becomes wider every day with the adoption of more powerful personal devices such as smartphones and netbooks.

A key aspect of the Funambol platform is that it is based on an open standard. SyncML is the standard protocol defined and maintained by the **Open Mobile Alliance** that is, **OMA**, (`http://www.openmobilealliance.org`) that addresses the problem of data synchronization between different devices and systems.

The full set of specification documents can be found at `http://www.openmobilealliance.org/Technical/current_releases.aspx`, under the **Data Synchronization** section as shown in the following screenshot:

OMA Enabler Releases

List of Enablers	Release Type	OMA Phase 1: Candidate Release	OMA Phase 2: Approved Release
OMA Application Layer Security Common Functions	Enabler Release	-	V1.0
OMA Browsing	Enabler Release	V2.4	V2.3 V2.2 V2.1
OMA Browser Protocol Stack	Enabler Release	V2.1	-
OMA Categorization Based Content Screening Framework	Enabler Release	V1.0	-
OMA Charging	Enabler Release	V1.1	V1.0
OMA Charging Data	Reference Release	V1.0	-
OMA Client Provisioning	Enabler Release	-	V1.1
OMA Client Side Content Screening Framework	Enabler Release	-	V1.0
OMA Condition Based URIs Selection **UPDATED 2009-10-20**	Enabler Release	**V1.0**	-
OMA Content Management Interface **UPDATED 2009-10-20**	Enabler Release	**V1.0**	-
OMA Converged Address Book **UPDATED 2009-10-20**	Enabler Release	**V1.0**	-
OMA Converged IP Messaging	Enabler Release	V1.0	-
OMA Customized Multimedia Ringing **UPDATED 2009-10-20**	Enabler Release	**V1.0**	-
OMA Data Objects	Reference Release	-	V1.0
OMA Data Synchronization	Enabler Release	V2.0	V1.2.2 V1.2.1 V1.1.2
OMA Device Capability Management Object **NEW ETS & EVP 2009-10-20**	Enabler Release	**V1.0**	-

It is worth noting that SyncML not only groups the specifications for data synchronization as seen until now, but also covers the so called SyncML Device Management, which will be briefly introduced in this chapter.

The SyncML initiative

In December 2000, Ericsson, IBM, Lotus, Motorola, Nokia, Palm, Psion, and Starfish Software formed the SyncML initiative to accelerate the market's vision of ubiquitous data access from any networked device. Their goal was to create a common synchronization protocol that could be used industry-wide. They worked with end users, device manufacturers, data providers, infrastructure developers, application developers, and service providers to define a common mobile data synchronization protocol that would satisfy the resource constraints of mobile devices and wireless networks and still provide the extensibility to support a range of applications and data types. The result was SyncML.

In 2002, the original SyncML initiative was consolidated into the Open Mobile Alliance under the **Data Synchronization Working Group** (**OMA DS**) and **Device Management Working Group** (**OMA DM**). OMA is the leading industry forum for developing market-driven, interoperable mobile service enablers. OMA was formed in June 2002 by nearly 200 companies including the world's leading mobile operators, device and network suppliers, information technology companies, and content and service providers.

The mission of the Open Mobile Alliance is to facilitate global user adoption of mobile data services by specifying market-driven mobile service enablers that ensure service interoperability across devices, geographies, service providers, operators, and networks, while allowing businesses to compete through innovation and differentiation.

The SyncML protocol

SyncML Data Synchronization is defined as the process of making two sets of data look identical. To achieve this goal, starting from a coherent initial state (for example, after synchronization) the two sets of data need to be modified accordingly and a conflict resolution policy must be followed. A conflict arises when the same data element is modified in both sets in an inconsistent way.

What is a synchronization protocol? First of all, it is a way to communicate the modifications between different datasets over a period of time.

The primary characteristics of a synchronization protocol are as follows:

- The addressing of the datasets (or database) and single records (in order to identify a particular database or a single record in a database)
- The definition of the standard commands needed to communicate modifications
- The definition of the ways these modifications can be exchanged (required features, transport protocols, and so on)

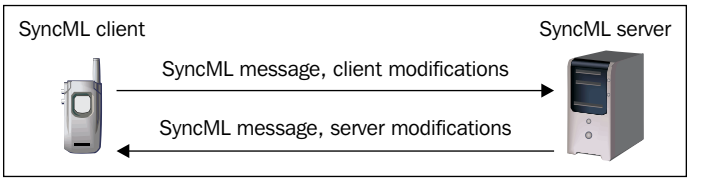
SyncML is an initiative to develop and promote a single, common data synchronization protocol that can be used industry-wide. Moreover, SyncML is a specification for a common data synchronization framework based on the exchange of XML-based messages between networked devices. SyncML is sponsored by companies such as Motorola, Nokia, Ericsson, IBM, and many others.

The SyncML specifications include:

- A representation protocol, which defines the XML messages format
- A synchronization protocol, which defines how SyncML messages shall be combined together in order to accomplish a synchronization session
- The binding for different transport protocols (HTTP, OBEX)
- A protocol for device management

SyncML client and server

The following figure shows a simple case where a mobile phone acts as a client connecting to a server.



The phone transmits a SyncML message to the server, including the most recent modifications. The server synchronizes its dataset (based on the changes received by the client, in terms of commands such as `Add`, `Delete`, and `Replace`) and responds with a SyncML message including its own modifications. This is a trivial example, but is useful to show the role assumed by the two devices during synchronization.

The **SyncML client** contains a synchronization agent and typically sends its modifications first. The **SyncML server** is the device implementing both the server-side synchronization agent and the synchronization logic, including the procedures to interpret the modifications, to discover and manage conflicts, and to generate return messages.

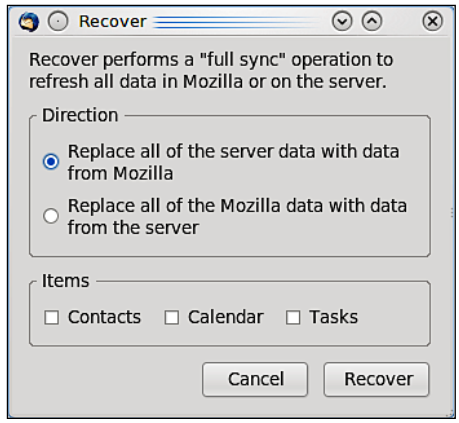
Synchronization modes

SyncML defines seven synchronization modes:

1. **Two-way sync**: The most common synchronization mode, where client and server exchange modifications on their data. The client first sends the modifications and the server then returns its own.
2. **Slow sync/full sync**: This is a special case of two-way sync where all the client database records are compared, field by field, with the ones on the server. The client sends the full database to the server, which in turn analyzes all records detecting missing items that need to be sent back to the client. It is typically used to recover from a situation where it is impossible to resolve synchronization conflicts and to reset to a consistent state between the client and the server. Another case when slow sync could be used is when client and server detect that they are out of sync so that a correct state must be recreated from scratch.
3. **One-way sync from the client only**: The client sends its modifications to the server, but no records from the server are expected nor returned.
4. **Refresh sync from client only**: The client sends its entire database to the server, which replaces its database with the data sent from the client.
5. **One-way sync from the server only**: The server sends its modifications to the client, but no client modifications are expected in return.
6. **Refresh sync from server only**: The server sends its entire database to the client. The client replaces the local set of data with the received one.
7. **Server alerted sync**: The server sends a synchronization alert to the client notifying it of the synchronization mode to be used.

It is not necessary for all clients to implement and support all these types of synchronization. For instance, the Funambol Mozilla Sync Client (described in Chapter 1) implements fast sync, full sync, refresh from client, and refresh from server. A server, however, is required to implement all synchronization types.

With the Funambol Mozilla Sync Client, fast and slow syncs are selected automatically by the client, based on the status of the synchronization between the client and the server. Refresh syncs can instead be launched from the application menu selecting **Tools | Recover.** The following screenshot will be displayed and the user can choose whether to restore the server or the client database.



SyncML basics

In the following sections, we will consider the basic elements needed to understand data synchronization in general and SyncML synchronization in particular. The following concepts will be presented:

- Sync anchors
- ID mapping

- Conflicts
- Security
- Addressing
- Devices and services addressing
- Databases addressing
- Item addressing
- Device capabilities

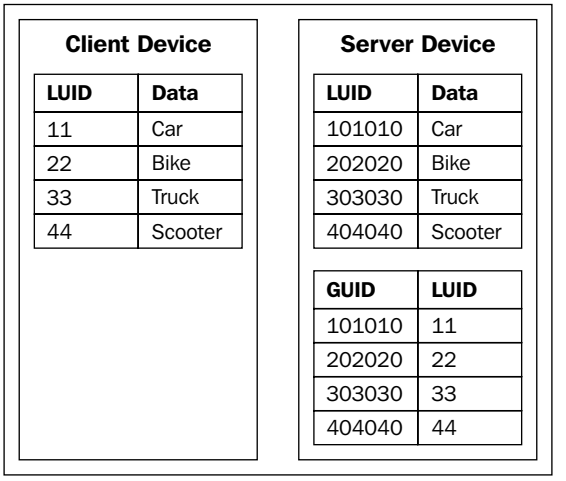
Sync anchors

In order to allow a sanity check on the synchronization state between two devices (was the last synchronization successfully completed?), SyncML makes use of two synchronization anchors, `Next` and `Last`. Each database *owns* the anchors which are sent to the other device during the synchronization initialization.

`Last` represents the last synchronization successfully performed. Usually it is a timestamp, but it could be a different identifier such as a session identifier. `Next` is a tag representing the current synchronization. Client and server exchange their anchors at the beginning of the synchronization process and are required to store the next anchor of the counterpart at the end of the sync process. Using this piece of information the server can detect if the last synchronization encountered problems and did not end correctly. For example, if the given `Last` is equal to the stored anchor, then the last synchronization was successfully terminated and, if the anchors do not match, client and server are out of sync and have to take actions to solve the issue. For this to work, it is important that the `Next` anchors are saved into the local repository only at the end of a successful synchronization.

ID mapping

In a multidevice synchronization scenario, the client and server must be able to create and deal with local item IDs independently. As a result, the server and client may have a different identifier for the same item while the server has to keep a mapping table between the client's **Locally Unique IDs** (**LUID**) and the server's **Globally Unique IDs** (**GUID**) as shown in the following figure:



Note that LUIDs are always created by the client device and only later communicated to the server with the SyncML command `map`.

Conflicts

A modification conflict arises when the same item has been modified on both server and client. When this happens, the synchronization engine has the responsibility of resolving the conflict. The client application is notified of the error condition by means of a status code.

For example, the following message represents a case where the server is notifying the client about a conflict resolution:

```
<Status>
  <CmdID>1</CmdID>
  <MsgRef>1</MsgRef>
  <CmdRef>2</CmdRef>
  <Cmd>Replace</Cmd>
  <SourceRef>1212</SourceRef>
  <Data>208</Data> <!-- Conflict, originator wins -->
</Status>
```

Security

SyncML mandates two authentication methods—basic and MD5.

Addressing

Addressing defines the way in which entities involved in the synchronization (databases, items, and so on) can be addressed. This is achieved by a naming convention which is defined on the basis of URIs.

For example, a typical server addressing URI might be:

```
<Source>
  <LocURI>http://sync.syncml.org/sync-server</LocURI>
</Source>
```

A client could use a different addressing identifier such as its `IMEI` number:

```
<Source>
  <LocURI>IMEI:493005100592800</LocURI>
</Source>
```

Also, the dataset to be synchronized is addressed with a URI. Note that the URI can be either absolute or relative.

```
<Sync>
  <Target>
    <LocURI>./calendar</LocURI>
  </Target>
</Sync>
```

```
<Sync>
  <Target>
    <LocURI>http://www.syncml.org/sync/calendar</LocURI>
  </Target>
</Sync>
```

Finally, even a single item of a database can be identified with the same method.

```
<Item>
  <Source>
    <LocURI>101</LocURI>
  </Source>
</Item>
```

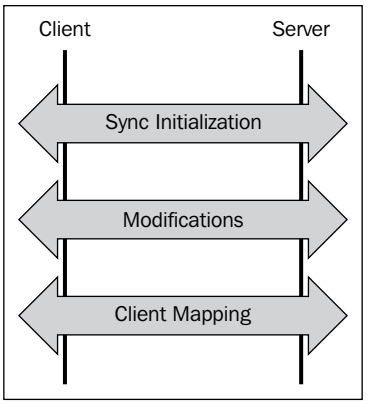
Device capabilities

SyncML defines how the devices can exchange their capabilities during the initialization phase. Note that the synchronizing devices are not required to send their capabilities unless requested by the counterpart. The exchange of device capabilities is very useful at the server side, as the server can apply proper optimization according to the remote device resources and features (database types, available memory, and transmission speed).

One thing to consider in this context is that capabilities information can be quite large. For this reason, the device's capabilities should be sent only if requested. In addition, the information should conform to the capabilities of the counterpart. For example, if the client notifies the server that it does not support the data format vCard3.0, the server should not insert in its response, the vCard3.0 properties it supports.

SyncML synchronization

The complete SyncML synchronization process can be represented as a sequence of phases, as shown in the following figure:



This is a three-step sequence—initialization, modifications exchange, and ID mapping.

Each step of the synchronization process is a packet, which is composed of multiple messages. This is necessary in order to meet particular requirements of the transport protocol or to support connection and device limitations.

SyncML specifies that before sending data modifications, the synchronizing counterparts exchange an initialization packet specifying the device characteristics and the synchronization requirements (for example, which databases need to be synchronized, which type of sync is needed, or which protocol features are supported). Note that the initialization packet can be merged with the modification package thereby reducing the chattiness of the communication. The drawback is that the synchronization process cannot be optimized as it could be, if the information capabilities were available before performing the synchronization analysis. The initialization package may also contain authentication credentials.

Each SyncML message is a well formed XML document with the element `<SyncML>` as root element and container for all the other tags. A single message is represented by a header within the element `<SyncHdr>` and a body enclosed in a `<SyncBody>` element as shown in the following image:

```
<SyncML>
<SyncHdr>
...
</SyncHdr>
<SyncBody>
...
</SyncBody>
</SyncML>
```

Note that as the XML message is not validated, the XML declaration and prolog can be omitted. The header contains routing information, database addressing, and protocol versioning, while the body contains one or more synchronization commands, as illustrated in the following image:

```
<SyncHdr>
<VerDTD>1.1</VerDTD>
<VerProto>SyncML/1.1</VerProto>
<SessionID>GWQKAKA</SessionID>
<MsgID>msg21</MsgID>
...
</SyncHdr>
```

Each command is represented by a specific XML element, which can be a container for subelements, data items, or meta information.

```
<SyncBody>
<Add>
<CmdID>cmdABC</CmdID>
...
<Item>...</Item>
</Add>
</SyncBody>
```

SyncML commands can be divided into **Request commands** and **Response commands**.

Request commands include `Add`, `Alert`, `Atomic`, `Copy`, `Delete`, `Exec`, `Get`, `Map`, `Put`, `Replace`, `Search`, `Sequence`, and `Sync`. Response commands include `Status` and `Results`.

Note that the protocol does not specify any semantics for these operations and, as a result, the `Add` operation could have a different meaning on a different device.

The following table gives a brief description of each command:

Command	Description
`Add`	Adds items to a database
`Alert`	Notifies the counterpart of the intention to synchronize a particular database
`Atomic`	All items included within an `Atomic` command must either "all succeed" or "all fail"
`Copy`	Copies the items into the destination database
`Delete`	Deletes the included items from the destination database

Command	Description
`Exec`	Executes a program on the destination device
`Get`	Requests information from the remote device
`Map`	Updates LUID-GUID mapping
`Put`	Sends information (such as capabilities) to the device
`Replace`	Replaces the included items into the destination database
`Search`	Specifies a search on the other device
`Sequence`	The included commands must be executed sequentially
`Sync`	The container for the modification commands
`Status`	Status notification for the execution and interpretation of any other command
`Results`	Contains `Get` or `Search` results

A Synchronization example

A two-way sync looks like the following sequence of messages:

```
<SyncML>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>SyncML/1.1</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target>
      <LocURI>http://www.syncml.org/sync-server</LocURI>
    </Target>
    <Source><LocURI>IMEI:493005100592800</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
```

```
<Status>
  <CmdID>1</CmdID>
  <MsgRef>1</MsgRef><CmdRef>0</CmdRef>
                    <Cmd>SyncHdr</Cmd>
  <TargetRef>IMEI:493005100592800</TargetRef>
  <SourceRef> http://www.syncml.org/
              sync-server </SourceRef>
  <Data>212</Data>
  <!--Statuscode for OK, authenticated for session-->
</Status>
<Status>
  <CmdID>2</CmdID>
  <MsgRef>1</MsgRef><CmdRef>5</CmdRef>
                    <Cmd>Alert</Cmd>
  <TargetRef>./dev-contacts</TargetRef>
  <SourceRef>./contacts/james_bond</SourceRef>
  <Data>200</Data>
  <!--Statuscode for Success-->
  <Item>
    <Data>
      <Anchor xmlns='syncml:metinf'>
        <Next>200005022T093223Z </Next>
      </Anchor>
    </Data>
  </Item>
</Status>
<Sync>
  <CmdID>3</CmdID>
  <Target><LocURI>./contacts/james_bond
          </LocURI></Target>
  <Source><LocURI>./dev-contacts</LocURI></Source>
  <Meta>
    <Mem xmlns='syncml:metinf'>
      <FreeMem>8100</FreeMem>
     <!--Free memory (bytes) in Calendar
         database on a device -->
      <FreeId>81</FreeId>
      <!--Number of free records in Calendar
         database-->
```

```
          </Mem>
        </Meta>
        <Replace>
          <CmdID>4</CmdID>
          <Meta><Type xmlns='syncml:metinf'>text/x-vcard
          </Type></Meta>
          <Item>
            <Source><LocURI>1012</LocURI></Source>
            <Data>
             <!--The vCard data would be placed here.-->
            </Data>
          </Item>
        </Replace>
      </Sync>
      <Final/>
    </SyncBody>
  </SyncML>
```

The request header contains the version number of the protocol, the DTD utilized, the session and message IDs, and the addressing of the target server (`Target` indicates the destination server and `Source` the client identifier).

The body starts with a `<Status>` command (note that the reported message is a modification message, we are assuming that the initialization already took place).

The next `<Status>` command responds to an `<Alert>` command requesting the synchronization of the server database `./contact/james_bond`. Note the use of the `next` anchor as described in the earlier section *Sync anchors*.

The next command in the message is a `<Sync>` command containing the synchronization operations to perform. `<Target>` and `<Source>` specify the databases, while the `<Meta>` command includes information regarding the client status.

Finally, the command `<Replace>` represents the required operation (replace the inner items with the data contained in `<Data>`). Note that the `<Meta>` element specifies the data type of the data to be stored.

The following code is an example of a response:

```
<SyncML>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>SyncML/1.1</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target><LocURI>IMEI:493005100592800</LocURI>
    </Target>
    <Source><LocURI>http://www.syncml.org/sync-server
    </LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <CmdID>1</CmdID>
      <MsgRef>2</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr
      </Cmd>
      <TargetRef>http://www.syncml.org/sync-server
      </TargetRef>
      <SourceRef>IMEI:493005100592800</SourceRef>
      <Data>200</Data>
    </Status>
    <Status>
  <!--This is a status for the client modifications to
      the server.-->
      <CmdID>2</CmdID>
      <MsgRef>2</MsgRef><CmdRef>3</CmdRef><Cmd>Sync</Cmd>
      <TargetRef>./contacts/james_bond</TargetRef>
      <SourceRef>./dev-contacts</SourceRef>
      <Data>200</Data> <!--Statuscode for Success-->
    </Status>
```

```
<Status>
  <CmdID>3</CmdID>
  <MsgRef>2</MsgRef>
  <CmdRef>4</CmdRef><Cmd>Replace</Cmd>
  <SourceRef>1012</SourceRef>
  <Data>200</Data> <!--Statuscode for Success-->
</Status>
<Sync>
  <CmdID>4</CmdID>
  <Target><LocURI>./dev-contacts</LocURI></Target>
  <Source><LocURI>./contacts/james_bond</LocURI>
  </Source>
  <Replace>
    <CmdID>5</CmdID>
    <Meta>
     <Type xmlns='syncml:metinf'>text/x-vcard</type>
    </Meta>
    <Item>
      <Target><LocURI>1023</LocURI></Target>
      <Data>
       <!--The vCard data would be placed here.-->
      </Data>
    </Item>
  </Replace>
  <Add>
    <CmdID>6</CmdID>
    <Meta>
     <Type xmlns='syncml:metinf'>text/x-vcard</type>
    </Meta>
    <Item>
      <Source><LocURI>10536681</LocURI></Source>
      <Data>
       <!--The vCard data would be placed here.-->
      </Data>
    </Item>
```

```
        </Add>
      </Sync>
      <Final/>
    </SyncBody>
  </SyncML>
```

The structure of the message is similar to the request message. Note the `<Status>` command with `<CmdId>` equals to `2`: It is the status of the synchronization process; it has a value `200` in our case, which indicates that the synchronization completed successfully. The server modifications (one `<Replace>` and one `<Add>`) are embedded in the `<Sync>` command addressed to the database: `./dev-contacts`.

The `<Final>` command indicates that this is the last message in the packet. The server therefore expects no further messages from the client.

SyncML device management

Configuring a mobile device is a difficult task, even for an expert user. Network parameters change from phone to phone, visual interfaces are difficult to use and understand, and all mobile operators have different configurations. Moreover, mobile devices are becoming *smarter* every day. New features are prepackaged on the device and new applications are downloaded and installed by the user. As a consequence, the task of configuring a mobile device is getting even more complex. Consumers, Corporate Information Management (IM/IT) departments, and operators are looking for a platform that allows remote management of devices in a convenient and effective way.

To answer this pressing need, some companies have started offering device management implementations for the mobile world. However, the available solutions are all based on different proprietary implementations. A proprietary platform exposes end users, mobile operators, device manufacturers, and Enterprise/IT departments to a

serious risk of insufficient interoperability and associated cost implications. In the long run, it means being locked in with a vendor and a technology, while the market is moving towards adopting new standards.

The OMA Device Management protocol is an open, universal industry standard for remote device management of networked devices. It is a widely adopted protocol that is nowadays available on majority of the mobile phones in the market (even if the service can be hidden to the user or not implemented by carriers).

Device management is a generic term used for a technology that allows third parties to carry out the difficult procedure of configuring mobile devices on behalf of the end user. Third parties would typically be wireless operators, service providers, or corporate information management departments.

Through device management, an external party can remotely set parameters, troubleshoot service problems, and install or upgrade software.

In broad terms, device management consists of three parts:

- **Protocol and Mechanism**: The protocol used between a management server and a mobile device.
- **Data model**: The data made available for remote manipulation, for example, browser and mail settings.
- **Policy**: The policy specifying who can manipulate a particular parameter or update a particular object in the device.

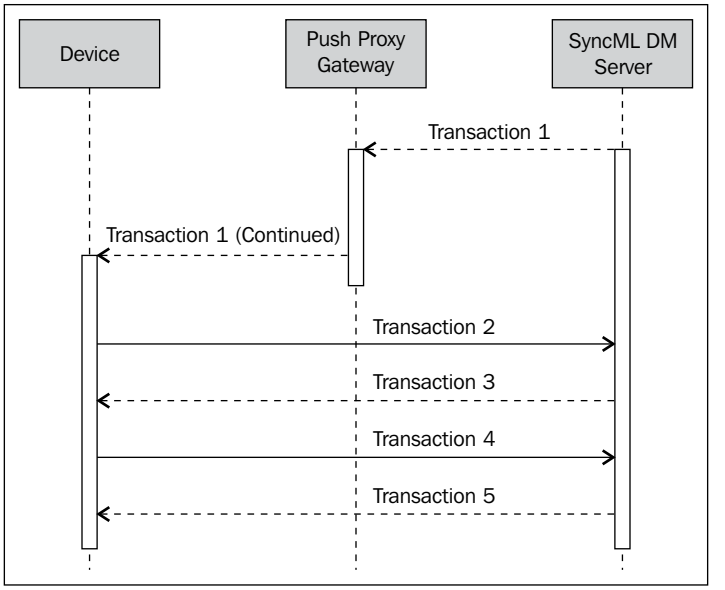
In a wireless environment, the crucial element for a device management protocol is the need to efficiently and effectively address the characteristics of mobile devices, including low bandwidth and high latency. The OMA DM protocol has been built keeping these requirements in mind.

OMA DM device management scope includes device configuration (modifying or reading operating parameters), software maintenance, inventory (reading from a device its current operating parameters, listing installed or running software, determining hardware configurations), and diagnostics (listening for alerts sent from a device, invoking local diagnostics on a device).

OMA DM protocol

The OMA DM protocol is relatively simple from the standpoint of messaging sequence. The message sequence is essentially broken into three parts, as illustrated in the following diagram:

- **Alert phase**: Used only for server-initiated management sessions
- **Setup phase**: Authentication and device information exchange
- **Management phase**: Device management commands exchange (this is where the management activity takes place)



Transaction 1: Alert phase

This phase is optional and data flows only from server to client. The server sends a notification package to the client requesting it to start a new management session. This is usually done with a WAP push message sent by a Push Proxy Gateway on behalf of the DM server that acts as the WAP Push initiator agent.

Transaction 2: Setup phase (from client)

This phase is always required and data flows from client to server. It consists of a request from the client and the response from the server. The initial client request contains three primary pieces of information:

- **Device information**: Data such as device ID, manufacturer, model tag, phone language, and DM protocol version
- **Client credentials**: Used for authentication purposes
- **Session alert**: Specifies whether the incoming session is client or server initiated

Transaction 3: Setup phase (from server)

The server responds to the initial client request with server credentials with the goal of identifying the server to the client for authentication and identification purposes. The server also sends user interaction and the first management command with the response.

Transactions 4 and 5: Device management

These two transactions represent the core of the management session. If the server sends a management command to the client in Transaction 3, then the client responds with data and status. Afterwards, the server can issue new management commands or simply return status information. The management session ends when the server sends neither additional management nor user interaction commands.

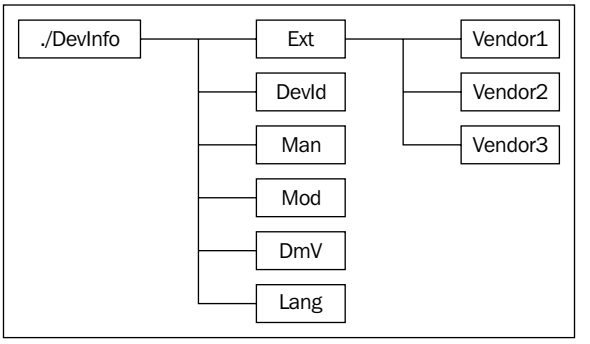
Device management tree

Device configuration data is organized in a hierarchical structure called the **device management tree**. Subtrees of the device management tree are called device management nodes, and a leaf, usually a single configuration parameter, is called a **manageable object**. Device objects can be anything from a single parameter, to a splash screen GIF file, to an entire application.

The device management tree is essentially mapped to permanent or dynamic objects as an addressing schema to manipulate these objects. Permanent objects can be thought of as objects that are built into the device at the time of manufacturing and typically cannot be deleted. Dynamic objects are objects that can be added or deleted (for example, network parameters, ring tones, or wallpapers).

The device management node ./DevInfo

As previously mentioned, the initial request from the client always contains device information whereby the data is retrieved from the **./DevInfo** subtree. The ./DevInfo node is only part of the overall device management tree structure, and it maps to basic device parameters that will allow initial operations and inspection of the device by the customer support specialists. The `./DevInfo` object is a standardized object so that any compliant device exposes the same basic information following the structure illustrated in the following figure:



The management subtree in the figure is a good example of how management objects are organized. ./DevInfo is the main node that includes subtrees (other management nodes) and leaves (manageable objects).The preceding figure shows the following subtrees:

- **Ext** is a subtree where any vendor implementation can expose vendor specific information
- **DevId** contains the device ID (such as the IMEI code for a phone device)
- **Man** is the device manufacturer identifier
- **Mod** represents the device model identifier
- **DmV** specifies the OMA DM client version
- **Lang** is the current language setting of the device

In addition to the manageable object value, each manageable object has a set of properties associated with it. These properties define metadata information about an object to allow things such as access control. These properties are listed in the following table:

Property	Description
`Access Control List` (ACL)	Defines who can manipulate the underlying object (required)
`Format`	Specifies how an object should be interpreted. For example, if the underlying object is a URL for a particular management server then the `Format` may be defined as `chr` or `character` (required)
`Name`	The name of the object in the tree (required)
`Size`	The size of the object in bytes (required for leaf objects, not applicable for interior nodes)
`Title`	User-friendly name of the object (optional)
`Tstamp`	The timestamp of the last modification (optional)

Property	Description
`Type`	The MIME type of the object (required for leaf objects, optional for Interior Nodes)
`VerNo`	The version number of the object (optional)

Device management commands

Management objects can be manipulated via SyncML messages with the following commands:

- `Add`: Adds an object (node) to a tree
- `Get`: Returns a node name based on the URI passed with the `GET` request
- `Replace`: Replaces an object on the tree
- `Delete`: Deletes an object on the tree
- `Copy`: Copies an object on the tree
- `Exec`: Executes a device-defined command on an object on the tree

Funambol Device Management server

Funambol has an implementation of a SyncML Device Management server. It is a community project so that Funambol staff is not directly involved in the maintenance and development of the project. The project can be found on the Funambol forge at the URL `https://dm.forge.funambol.org`.

Summary

In this chapter, the SyncML protocol was described in detail. SyncML specifications cover two main areas of mobile applications—data synchronization and device management. The book is focused on the former protocol and applications, but for the sake of completeness it is worth learning more about SyncML DM, for which Funambol hosts a community implementation of the server.

10
Extending the Funambol Data Synchronization Service

The previous chapter covered the SyncML protocol in detail. Funambol uses SyncML as its synchronization protocol to exchange data between the client and server.

This chapter explains how to develop extensions for the Funambol DS Service so that a developer can create integrations with other backend datasources. It will also describe how to build a simple connector, helping us to understand how the Funambol DS Service interacts with server extensions and connectors.

Funambol development

The following sections present several concepts related to how Funambol can be used to build mobile applications. Before digging into the details of Funambol development, it is useful to describe some basic concepts of data synchronization, as it is the foundation for Funambol applications and services. Some of these concepts such as the SyncML protocol were introduced in the previous chapter. In this chapter, those aspects will be described from a more technical perspective.

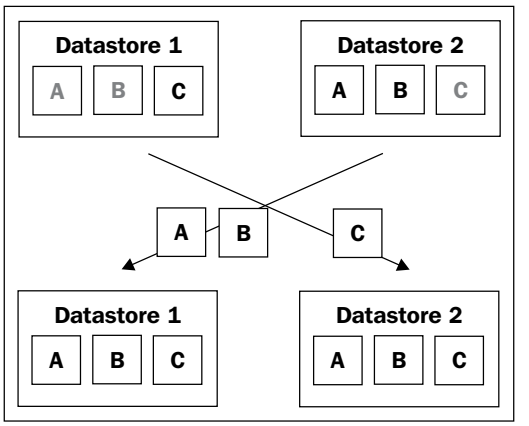
Data synchronization

All mobile devices—handheld computers, mobile phones, pagers, and laptops—need to synchronize their data with the server where the information is stored. This ability to access and update information on the fly is the key to the pervasive nature of mobile computing. Yet, today almost every device uses a different technology for data synchronization.

Data synchronization is helpful for:

- Propagating updates between a growing number of applications
- Overcoming the limitations of mobile devices and wireless connections
- Maximizing the user experience by minimizing data access latency
- Keeping scalability of the infrastructure in an environment where the number of devices (clients) and connections tend to increase considerably
- Understanding the requirements of mobile applications, providing a user experience that is helpful, and not an obstacle, for mobile tasks

Data synchronization is the process of making two sets of data look identical, as shown in the following figure:



This involves many techniques, which will be discussed in the following sections. The most important are:

- ID handling
- Change detection
- Modification exchange
- Conflict detection
- Conflict resolution
- Slow and fast synchronization

ID handling

At first glance, ID handling seems like a pretty straightforward process that requires little or no attention. However, ID handling is an important aspect of the synchronization process and is not trivial.

In some cases a piece of data is identifiable by a subset of its content fields. For example, in the case of a contact entry, the concatenation of a first name and last name uniquely selects an entry in the directory. In other cases, the ID is represented by a particular field specifically introduced for that purpose. For example, in a Sales Force Automation mobile application, an order is identified by an order number or ID. The way in which an item ID is generated is not predetermined and it may be application or even device specific.

In an enterprise system, data is stored in a centralized database, which is shared by many users. Each single item is recognized by the system because of a unique global ID. In some cases, two sets of data (the order on a client and the order on a server) represent the same information (the order made by the customer) but they differ. What could be done to reconcile client and server IDs to make the information consistent? Many approaches can be chosen:

- Client and server agree on an ID scheme (a convention on how to generate IDs must be defined and used).
- Each client generates globally unique IDs (GUIDs) and the server accepts client-generated IDs.
- The server generates GUIDs and each client accepts those IDs.
- Client and server generate their own IDs and a mapping is kept between the two. Client-side IDs are called Locally Unique Identifiers (LUID) and server-side IDs are called Globally Unique Identifiers (GUID). The mapping between local and global identifiers is referred as LUID-GUID mapping. The SyncML specifications prescribe the use of LUID-GUID mapping technique, which allows maximum freedom to client implementations.

Change detection

Change detection is the process of identifying the data that was modified after a particular point in time that is, the last synchronization. This is usually achieved by using additional information such as timestamp and state information. For example, a possible database enabled for efficient change detection is shown in the following table:

ID	First name	Last name	Telephone	State	Last_update
12	John	Doe	+1 650 5050403	N	2008-04-02 13:22
13	Mike	Smith	+1 469 4322045	D	2008-04-01 17:32
14	Vincent	Brown	+1 329 2662203	U	2008-03-21 17:29

However, sometimes legacy databases do not provide the information needed to accomplish efficient change detection. As a result, the matter becomes more complicated and alternative methods must be adopted (based on content comparison, for instance). This is one of the most important aspects to consider when writing a Funambol extension, because the synchronization engine needs to know what's changed from a point in time.

Modification exchange

A key component of a data synchronization infrastructure is the way modifications are exchanged between client and server. This involves the definition of a synchronization protocol that client and server use to initiate and execute a synchronization session. In addition to the exchange modification method, a synchronization protocol must also define a set of supported modification commands. The minimal set of modification commands are as follows:

- `Add`
- `Replace`
- `Delete`

Conflict detection

Let's assume that two users synchronize their local contacts database with a central server in the morning before going to the office. After synchronization, the contacts on their smartphones are exactly the same. Let's now assume that they update the telephone number for "John Doe" entry and one of them makes a mistake and enters a different number. What will happen the next morning when they both synchronize again? Which of the two new versions of the "John Doe" record should be taken and stored into the server? This condition is called **conflict** and the server has the duty of identifying and resolving it.

Funambol detects a conflict by means of a synchronization matrix shown in the following table:

Database A → ↓ Database B	**New**	**Deleted**	**Updated**	**Synchronized/ Unchanged**	**Not Existing**
New	C	C	C	C	B
Deleted	C	X	C	D	X
Updated	C	C	C	B	B
Synchronized/ Unchanged	C	D	A	=	B
Not Existing	A	X	A	A	X

As both users synchronize with the central database, we can consider what happens between the server database and one of the client databases at a time. Let's call **Database A**, as the client database and **Database B,** as the server database. The symbols in the synchronization matrix have the following meaning:

- X: Do nothing
- A: Item A replaces item B
- B: Item B replaces item A
- C: Conflict
- D: Delete the item from the source(s) containing it

Conflict resolution

Once a conflict arises and is detected, proper action must be taken. Different policies can be applied. Let's see some of them:

- **User decides**: The user is notified of the conflict condition and decides what to do.
- **Client wins**: The server silently replaces conflicting items with the ones sent by the client.
- **Server wins**: The client has to replace conflicting items with the ones from the server.
- **Timestamp based**: The last modified (in time) item wins.
- **Last/first in wins**: The last/first arrived item wins.
- **Merge**: Try to merge the changes, at least when there is no direct conflict. Consider the case of a vcard, where two concurrent modifications have been applied to two different fields. There is a conflict at the card level, but the two changes can be merged so that both clients can then have a valid version of the card. This is the best example of the case when the change is not directly conflicting.
- **Do not resolve**.

Note that Funambol adopts a special merging policy that guarantees that the user does not lose data. The server always tries to merge if possible. When a conflict cannot be resolved with merging (for example, there are conflicting changes on the same field), the value in the last synchronization wins over the older synchronizations to meet the expectation of the user who is synchronizing. In this way, when the users who applied previous changes receive the new updates all devices will be in sync.

Synchronization modes: Full or fast

As seen in the previous chapter, there are many modes to carry out the synchronization process. The main distinction is between fast and full synchronization. **Fast synchronization** involves only the items changed since the last synchronization between two devices. Of course, this is an optimized process that relies on the fact that, the devices were fully synchronized at some point in the past; this way, the state at the beginning of the sync operation is well known and sound. When this condition is not met (for instance, the mobile device has been reset and lost the timestamp of the last synchronization), a full synchronization must be performed. In a **full synchronization**, the client sends its entire database to the server, which compares it with its local database and returns the modifications that must be applied to be up-to-date again.

Both fast and full synchronization modes can be performed in one of the following manners:

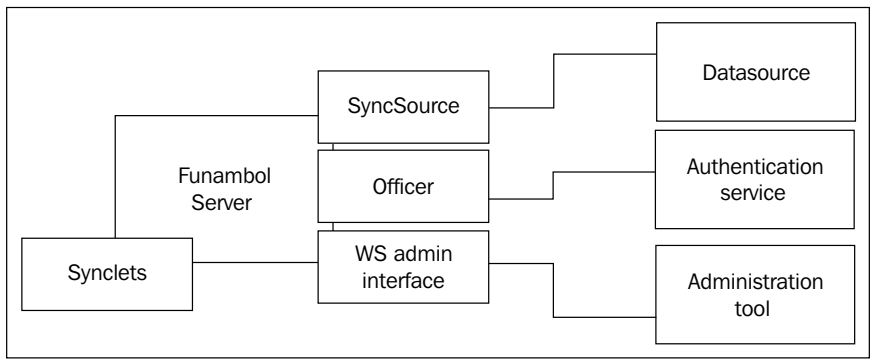
- **Client-to-server**: The server updates its database with client modifications, but sends no server-side modifications
- **Server-to-client**: The client updates its database with server modifications, but sends no client-side modifications
- **Two-way**: The client and server exchange their modifications and both databases are updated accordingly

Extending Funambol

The Funambol platform can be extended in many areas to integrate Funambol with existing systems and environments. The most common integration use cases and the Funambol modules involved are:

- **Officer**: Integrating with an external authentication and authorization service
- **SyncSource**: Integrating with an external datasource to address client specific issues
- **Synclet**: Adding pre or postprocessing to a SyncML message
- **Admin WS**: Integrating with an external management tool

These are illustrated in the following diagram:



Funambol extensions are distributed and deployed as Funambol modules. This section describes the structure of a Funambol module, while the following sections describe each of these listed scenarios.

A Funambol module represents the means by which developers can extend the Funambol server. A module is a packaged set of files containing Java classes, installation scripts, configuration files, initialization SQL scripts, components, and so on, used by the installation procedure to embed extensions into the server core.

For more information on how to install Funambol modules, see the *Funambol Installation and Administration Guide*.

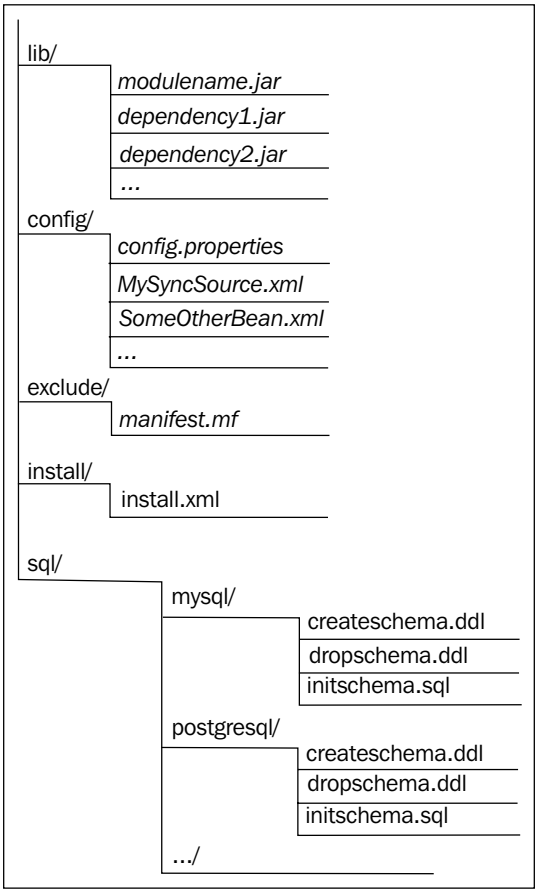
Building a Funambol module

A Funambol module is a ZIP package named using the following convention:

```
<module-name>-<major-version>.<minor-version>.s4j
```

Here `<module-name>` is the name of the module without spaces and using lowercase characters only, and `<major/minor-version>` are the major and minor version numbers. A new version of a module with a minor version number change must be backward compatible, while changes in the major version number imply that a migration may be required.

The package must have the structure illustrated in the following diagram:



In the previous diagram, entries ending with a '/' represent directories and filenames in italic are given as examples (in a real package they will be replaced with real filenames).

The module classes are packaged in a main JAR file called `<modulename>.jar`.

Configuration files are stored under the package directory `config`, creating subdirectories as needed.

Even if it is not mandatory, usually SyncSource instance configuration files are stored under a subtree in the form `<module-id>/<connector-id>/<sourcetype-id>`, which is the convention used by the Funambol Administration Tool when creating a new SyncSource instance.

The directory `install` contains `install.xml`, an Apache Ant script that is called when the module is installed; this is the hook where a module developer can insert module specific installation tasks. Installation specific files can be organized in subdirectories under `install`.

If the module requires a custom database schema, the scripts to create, drop, and initialize the database are stored under the `sql/<database>` directory, where `<database>` is the name of the DBMS as listed in the `install.properties` file.

Finally, the `exclude` directory is used to store files that will be temporarily used by the installation procedure, but will not be (automatically) copied. These can be used by the module installation script for any purpose.

Modules, connectors, listeners, and SyncSource types

As mentioned, a module is a container for anything related to one or more server extensions, which are used by the engine to communicate and integrate with external systems. These extensions are usually specific with backend to be integrated. A specific case of such an extension is when the main purpose is to connect to an external datasource and in this case the module is called a **connector**. In other words, a connector is an extension of the server, intended to support the synchronization of a particular datasource.

To access the datasource, the connector must provide a SyncSource type. A SyncSource type represents the template from which an instance of a SyncSource can be created. For example, the `FileSystemSyncSource` type made available by Funambol is the means used by the server to synchronize data stored in the file system. However, it does not represent a particular directory to synchronize. In order to synchronize a specific directory (for instance `/data/contacts`) a real SyncSource instance must be created and configured with the desired directory. You can think of a SyncSource type as a class and of a SyncSource as an instance.

An additional (but optional) component that a connector can provide is called **listener**. This component is in charge of detecting changes in the backend so that the server can trigger a device to synchronize the changes.

Funambol provides a module called foundation that contains basic functionality, libraries, and classes required by all custom modules. This module must not be removed from any Funambol installation.

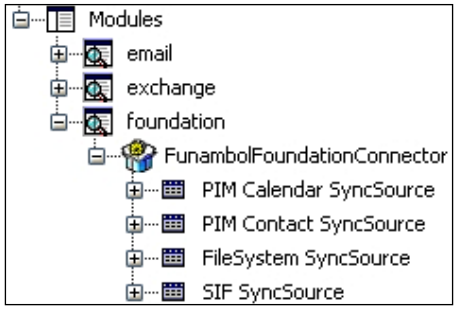
Registering modules, connectors, and SyncSource types

Modules, connectors, and SyncSource types are registered by filling the following database tables:

- `fnbl_module` for module information
- `fnbl_connector` for connector information
- `fnbl_sync_source_type` for SyncSource type information
- `fnbl_connector_source_type` for connector-SyncSource type associations
- `fnbl_module_connector` for module-connector association

The last two tables are used to create the hierarchy module-connector-SyncSource type that can be seen in the Funambol Administration Tool.

Let's consider the registration of the `foundation` module. When Funambol is installed, the `foundation` module is also installed. It brings a connector called `FunambolFoundationConnector` which, in turn, contains the SyncSource types `PIM Contact SyncSource`, `PIM Calendar SyncSource`, `FileSystem SyncSource`, and `SIF SyncSource` as shown in the following screenshot:



This hierarchy is obtained with the following SQL commands:

1. Module registration:

```
insert into fnbl_module (id, name, description)
values('foundation','foundation','Foundation');
```

2. SyncConnector registration:

```
insert into fnbl_connector(id, name, description)
values('foundation','FunambolFoundationConnector',
'Funambol Foundation Connector');
```

3. The Foundation Connector belongs to the foundation module:

```
insert into fnbl_module_connector(module, connector)
values('foundation','foundation');
```

4. The SyncSource Type registration:

```
insert into fnbl_sync_source_type(id, description,
class, admin_class)
values('contact-foundation','PIM Contact SyncSource',
'com.funambol.foundation.engine.source.
PIMContactSyncSource',
'com.funambol.foundation.admin.
PIMContactSyncSourceConfigPanel');
insert into fnbl_sync_source_type(id, description,
class,admin_class)
values('calendar-foundation','PIM Calendar
SyncSource',
'com.funambol.foundation.engine.source.
PIMCalendarSyncSource',
'com.funambol.foundation.admin.
PIMCalendarSyncSourceConfigPanel');
insert into fnbl_sync_source_type(id, description,
class, admin_class)
values('fs-foundation','FileSystem SyncSource',
'com.funambol.foundation.engine.source.
FileSystemSyncSource',
'com.funambol.foundation.admin.
FileSystemSyncSourceConfigPanel');
insert into fnbl_sync_source_type(id, description,
class, admin_class)
values('sif-fs-foundation','SIF SyncSource',
'com.funambol.foundation.engine.source.SIFSyncSource',
'com.funambol.foundation.admin.
SIFSyncSourceConfigPanel');
```

5. Finally, the SyncSource type belongs to the Foundation Connector:

```
insert into fnbl_connector_source_type(connector,
sourcetype)
values('foundation','contact-foundation');
insert into fnbl_connector_source_type(connector,
sourcetype)
values('foundation','calendar-foundation');
insert into fnbl_connector_source_type(connector,
sourcetype)
```

```
values('foundation','fs-foundation');

insert into fnbl_connector_source_type(connector,
 sourcetype)
values('foundation','sif-fs-foundation');
```

Two classes are specified for each SyncSource type registration—The class (for example, `com.funambol.foundation.engine.source.FileSystemSyncSource`) that actually implements the SyncSource interface, and the `admin_class` that is used to create a new SyncSource instance and to configure it in the Funambol Administration Tool.

In the following section, we will see how these two classes are developed.

In this guide, SyncSource and SyncSource Type are often treated as synonyms, even if they are in the template-instance relationship seen before.

Getting started on connector development

This section describes how to create a connector that extends the functionality of the Funambol Data Synchronization Service.

In this section we will use the following terms and concepts:

- **Module**: A module is a container for anything related to one or more server extensions, which are used by the engine to integrate with external systems.

- **Connector**: A connector is a particular type of module, whose primary purpose is to connect to an external datasource. In other words, a connector is an extension of the DS service intended to support the synchronization of a particular datasource.
- **SyncSource**: This is a key component of a connector that defines the way a set of data is made accessible to the Funambol Data Synchronization Service for synchronization. A SyncSource type represents the template from which an instance of a SyncSource can be created.
- **Synclet**: A preprocessing or postprocessing unit that can process a SyncML message before it gets into the synchronization engine or just after it is going out from it.

This chapter will guide the developer through the development, packaging, installation, and testing of a module. The module contains a simple SyncSource and Synclet, which will just produce some logging. In this way, the developer can easily see the flow of calls to the SyncSource API during a synchronization.

After becoming familiar with the tutorial, it is recommended that you inspect some real-world examples of connectors such as the OpenXchange connector (`https://funamboloxconnector.forge.funambol.org`) or the Exchange connector (`https://exchange-connector.forge.funambol.org`).

Getting started

The following connector development quick-start section assumes a working knowledge of Java, Maven, and SQL.

To follow this section you will need:

- Funambol Data Synchronization Service installed and running
- Funambol Software Development Kit
- Java 2 SDK version 1.5.x or above
- Apache Maven (see `http://maven.apache.org`)
- Optionally, you may want to download a Maven plugin for your preferred IDE (see `http://mevenide.codehaus.org`)

To download the Funambol SDK, go to `https://www.forge.funambol.org/download` and click on Funambol SDK in the **Get started with Funambol** section.

After downloading the software, install it in a directory of choice. In the following section we will use the below directory conventions:

- `$FUNAMBOL_HOME`: This is the directory where the bundle has been installed (for example, `/opt/Funambol`)
- `$FUNAMBOL_SDK_HOME`: This is the directory where the Funambol SDK has been installed (for example, `/opt/Funambol/tools/sdk`)
- `$JAVA_HOME`: This is the directory where Java is installed (for example, `/opt/jdk1.5.0_10`)
- `$MAVEN_HOME`: This is the directory where Apache Maven is installed (for example, `/opt/apache-maven-2.0.8`)
- `$USER_HOME`: This is the home directory of the operating system you are using (for example, `/home/ste`, `c:\Users\ste`)

Basic knowledge of Apache Maven, its terminology, and principles is assumed, as the following sections use terms from the Maven world without explaining them in detail. For more information refer to `http://maven.apache.org/guides/getting-started/index.html`.

After the installation, Maven should be configured to point to the Funambol public Maven repository (m2.funambol.org/repositories). To do so, copy the file `$FUNAMBOL_SDK_HOME/docs/settings.xml` under `$USER_HOME/.m2`.

Overview

The following sections are a guide on how to develop the sample module. This can be done using the following steps:

1. Create the connector project.
2. Install the module.
3. Create a SyncSource instance.
4. Test the module with a SyncML client.

Create the connector project

The easiest way to create a connector project is by running the following Maven command:

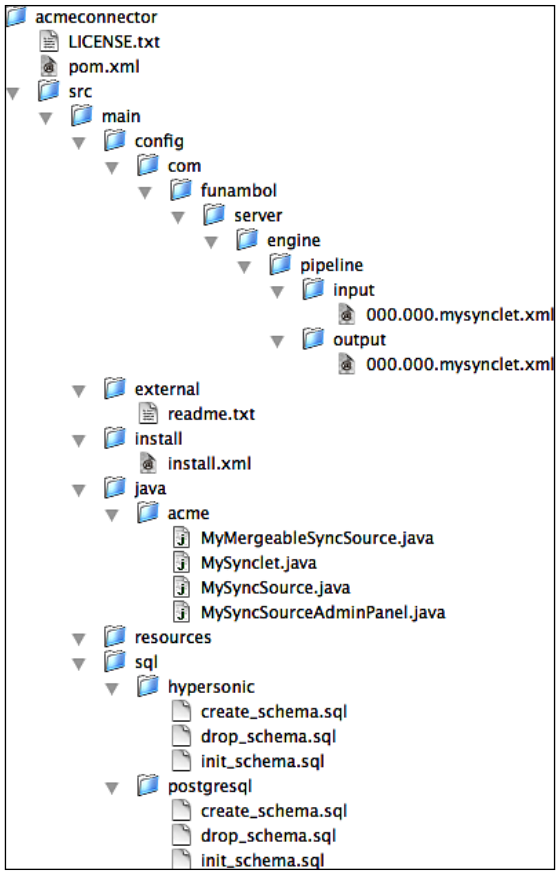
```
mvn archetype:generate -DarchetypeGroupId=funambol
-DarchetypeArtifactId=funambol-module-archetype -
DarchetypeVersion=7.1.0
-DgroupId=acme -DartifactId=acmeconnector
-DarchetypeRepository=http://m2.funambol.org/repositories/
artifacts
-Dversion=1.0.0
```

When prompted for an answer, type *Y*.

This command will download and create a skeleton application ready to be built, which contains:

- A normal SyncSource
- A mergeable SyncSource
- An input/output synclet
- All configuration and SQL files

All of these will be generated in a Maven project located in a directory named `acmeconnector`. The content of the directory is illustrated in the following screenshot:



The following table explains the function of each file:

File	Description
`LICENSE.txt`	AGPL license file
`pom.xml`	Maven project file for the connector
`src/main/config/com/funambol/server/engine/pipeline/input/000.000.mysynclet.xml`	Sample input synclet configuration
`src/main/config/com/funambol/server/engine/pipeline/output/000.000.mysynclet.xml`	Sample output synclet configuration
`src/main/external/readme.txt`	Readme for the content of this directory
`src/main/install/install.xml`	Module installation file
`src/main/java/acme/MyMergeableSyncSource.java`	Sample mergeable SyncSource
`src/main/java/acme/MySynclet.java`	Sample synclet
`src/main/java/acme/MySyncSource.java`	Sample SyncSource
`src/main/java/acme/MySyncSourceAdminPanel.java`	Sample administration panel for both MySyncSource and MyMergeableSyncSource
`src/main/sql/*/create_schema.sql`	SQL scrip to create the database tables required by the module
`src/main/sql/*/drop_schema.sql`	SQL scrip to drop the database tables required by the module
`src/main/sql/*/init_schema.sql`	SQL scrip to initialize the database tables required by the module

Take a moment to explore and open each file to become familiar with its contents.

MyMergeableSyncSource type

As seen earlier, the SyncSource type is the primary component of the connector. The source code created by the archetype is very simple. It only writes some logging information to trace its execution. However in a real project this is where the code necessary to integrate an external datasource would go.

`MyMergeableSyncSource` inherits most of its behavior from `MySyncSource`. Open this file in an editor or in your IDE and go through it. `MySyncSource` defines three properties that can be set via the Funambol Administration Tool. These are `myString`, `myInt`, and `myMap`. `Getter` and `setter` methods to get and set those properties are provided. Also, note that the class implements all methods of the SyncSource interface by writing a log entry. Each method also has a description of what it does and what the developer should add.

MySynclet

In addition to the SyncSource types described earlier, the archetype project also contains a sample input and output synclet—`MySynclet`. Open it in an editor or in your IDE and go through it.

If you don't need to write a synclet, you can skip this section.

The first thing to note is that it implements both an input and an output synclet, which means that the synclet will be called for both incoming and outgoing messages. The synclet is very simple; it just logs the message in the `funambol.myconnector` logger.

MySyncSourceAdminPanel

The goal of this section is to create a new SyncSource based on `MyMergeableSyncSource` and to configure it via the Funambol Administration Tool. This is possible because of the class `MySyncSourceAdminPanel`. Open it with an editor or in your IDE and go through it.

`MySyncSourceAdminPanel` inherits `SourceManagementPanel`, which is a class of the Admin framework. `SourceManagementPanel` is a `java.swing.JPanel`, therefore it has all the methods of a swing panel. The `init()` method creates all widgets that we want to display in the Funambol Administration Tool and adds them to the panel. These widgets are as follows:

- Source name
- Data types supported (e.g. text/vcard)
- Data type versions supported (e.g. 2.1)
- Source URI
- myString
- myInt
- myMap entry

In addition, it adds a JButton to the panel to save the values of an existing SyncSource or to add a newly created SyncSource. An important aspect to note is that this is where the panel interacts with the Funambol Administration Tool to save the changes to the server. The following code performs this task:

```
confirmButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent event ) {
    try {
      validateValues();
      getValues();
```

```
        if (getState() == STATE_INSERT)
        {
          SyncSourceAdminPanel.this.actionPerformed(
            newActionEvent(MySyncSourceAdminPanel.this,
                           ACTION_EVENT_INSERT,
                           event.getActionCommand()));
        } else {
          MySyncSourceAdminPanel.this.actionPerformed(
            new ActionEvent(MySyncSourceAdminPanel.this,
                            ACTION_EVENT_UPDATE,
                            event.getActionCommand()));
              }
      } catch (Exception e)
      {
        notifyError(new AdminException(e.getMessage()));
      }
    }
  });
```

The key is that, when needed, the method `actionPerformed()` of the base class is called with a proper event.

Another important method is `updateForm()` where the value of the SyncSource instance is displayed in proper fields. Again this method is called by the Funambol Administration Tool when an existing instance must be displayed.

Creating and installing the connector package

To package the created project into a Funambol module, go in the project directory and type:

```
mvn package
```

A typical output will be as follows:

```
[INFO] Scanning for projects...
[INFO] ------------------------------------------------
[INFO] Building acme acmeconnector Module
[INFO]    task-segment: [package]
[INFO] ------------------------------------------------
[INFO] artifact org.apache.maven.plugins:maven-resources-
plugin: checking for updates from artifacts
[INFO] artifact org.apache.maven.plugins:maven-resources-
plugin: checking for updates from snapshots
[INFO] artifact org.apache.maven.plugins:maven-compiler-
plugin: checking for updates from artifacts
[INFO] artifact org.apache.maven.plugins:maven-compiler-
plugin: checking for updates from snapshots
[INFO] artifact org.apache.maven.plugins:maven-surefire-
plugin: checking for updates from artifacts
[INFO] artifact org.apache.maven.plugins:maven-surefire-
plugin: checking for updates from snapshots
[INFO] artifact org.apache.maven.plugins:maven-jar-
plugin: checking for updates from artifacts
[INFO] artifact org.apache.maven.plugins:maven-jar-
plugin: checking for updates from snapshots
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Compiling 4 source files to /Users/ste/Projects/
acmeconnector/target/classes
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] No sources to compile
[INFO] [surefire:test]
[INFO] No tests to run.
```

```
[INFO] [jar:jar]
[INFO] Building jar: /Users/ste/Projects/acmeconnector/
target/acmeconnector-1.0-SNAPSHOT.jar
[INFO] [funambol:s4j]
[INFO] Exploding Funambol packaging...
[INFO] Assembling Funambol packaging acmeconnector in /
Users/ste/Projects/acmeconnector/target/acmeconnector-
1.0-SNAPSHOT
[INFO] Including license file /Users/ste/Projects/
acmeconnector/LICENSE.txt
[INFO]
[INFO] Including artifacts:
[INFO] -------------------
[INFO] x funambol:server-framework:jar:8.0.3-SNAPSHOT:
compile
[INFO] x funambol:core-framework:jar:6.5.4:compile
[INFO] x funambol:ext:jar:6.5.2:compile
[INFO] o org.jibx:jibx-run:jar:1.1.2fun:compile
[INFO] o xpp3:xpp3:jar:1.1.2a-fun:compile
[INFO] o commons-lang:commons-lang:jar:2.3:compile
[INFO] o funambol:admin-framework:jar:6.5.2:compile
[INFO]
[INFO] Excluded artifacts:
[INFO] -------------------
[INFO]
[INFO]     Including jar files...
[INFO] basedir: /Users/ste/Projects/acmeconnector
[INFO] srcDir: /Users/ste/Projects/acmeconnector/src/main
[INFO] sqlDirectory: /Users/ste/Projects/acmeconnector/
target/acmeconnector-1.0-SNAPSHOT/sql
[INFO] wsddDirectory: /Users/ste/Projects/acmeconnector/
target/acmeconnector-1.0-SNAPSHOT/wsdd
[INFO] No config files...
[INFO] No exclude files...
[INFO] Including install files...
```

```
[INFO] Including sql files...
[INFO] No wsdd files...
[INFO]
[INFO] Generating Funambol packaging /Users/ste/Projects/
acmeconnector/target/acmeconnector-1.0-SNAPSHOT.s4j
[INFO] Building jar: /Users/ste/Projects/acmeconnector/
target/acmeconnector-1.0-SNAPSHOT.s4j
[INFO] ------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------
[INFO] Total time: 12 seconds
[INFO] Finished at: Sat May 17 15:42:08 CEST 2008
[INFO] Final Memory: 9M/16M
[INFO] ------------------------------------------------
-----------
```

The final message `Generating Funambol packaging /Users/ste/Projects/acmeconnector/target/acmeconnector-1.0-SNAPSHOT.s4j` tells where the package is created.

To install the newly created module into the server, copy `acmeconnector-1.0-SNAPSHOT.s4j` into `$FUNAMBOL_HOME/ds-server/modules` and follow the steps detailed next:

1. Make sure Funambol is up and running.
2. Using a text editor, open the `$FUNAMBOL_HOME/ds-server/install.properties` file.
3. Find the line that begins with `modules-to-install=` in the Module definitions section. This line specifies, in a comma-separated list, the modules to install during installation.

4. Add `acmeconnector-1.0-SNAPSHOT` to the comma-separated list (note that you do not have to specify the `.s4j` file extension).
5. Save and close `install.properties`.
6. Open a command window and run the server installation script by typing:
   - on Windows:
     ```
     cd $FUNAMBOL_HOME/ds-server
     bin\install-modules
     ```
   - on Unix/Linux:
     ```
     cd $FUNAMBOL_HOME/ds-server
     bin/install-modules
     ```
7. Answer all questions about recreating the modules DB schema when prompted. Type *Y* when asked to install the database for the new module and *N* for all others.

```
[echo] Funambol Data Synchronization Server will be
installed on the Tomcat 5.5.x application server
[echo] Undeploying funambol...
[echo] Pre installation for modules foundation-
8.0.1,acmeconnector-1.0-SNAPSHOT
[echo] foundation-8.0.1 pre-installation...
[echo] foundation-8.0.1 pre-installation successfully
completed
[echo] acmeconnector-1.0-SNAPSHOT pre-installation...
[echo] acmeconnector-1.0-SNAPSHOT pre-installation
successfully completed
[echo] Copying configuration files
[echo] Post installation for modules foundation-
8.0.1,acmeconnector-1.0-SNAPSHOT
[echo] has.install: true
[echo] Starting custom installation...
[echo] Foundation Installation
[echo] Foundation installation successfully completed
[echo] foundation-8.0.1 installation...
```

```
[echo] Database installation for module foundation-
8.0.1 on hypersonic (/opt/Funambol/ds-server)

[iterate] The Funambol Data Synchronization Server
installation program can now create
[iterate] the database required by the module
foundation-8.0.1 (if any is needed).
[iterate] You can skip this step if you have already a
valid database created
[iterate] or the module does not require a database.
[iterate] If you choose 'y' your existing data will be
deleted.
[iterate] Do you want to recreate the database?
[iterate]       (y,n)
n

[...]

[echo] foundation-8.0.1 installation successfully
completed
[echo] has.install: true
[echo] Starting custom installation...
[echo] acmeconnector installation
[echo] acmeconnector installation successfully
completed
[echo] acmeconnector-1.0-SNAPSHOT installation...
[echo] Database installation for module acmeconnector-
1.0-SNAPSHOT on hypersonic (/opt/Funambol/ds-server)
[iterate] The Funambol Data Synchronization Server
installation program can now create
[iterate] the database required by the module
acmeconnector-1.0-SNAPSHOT (if any is needed).
[iterate] You can skip this step if you have already a
valid database created
[iterate] or the module does not require a database.
[iterate] If you choose 'y' your existing data will be
deleted.
[iterate] Do you want to recreate the database?
```

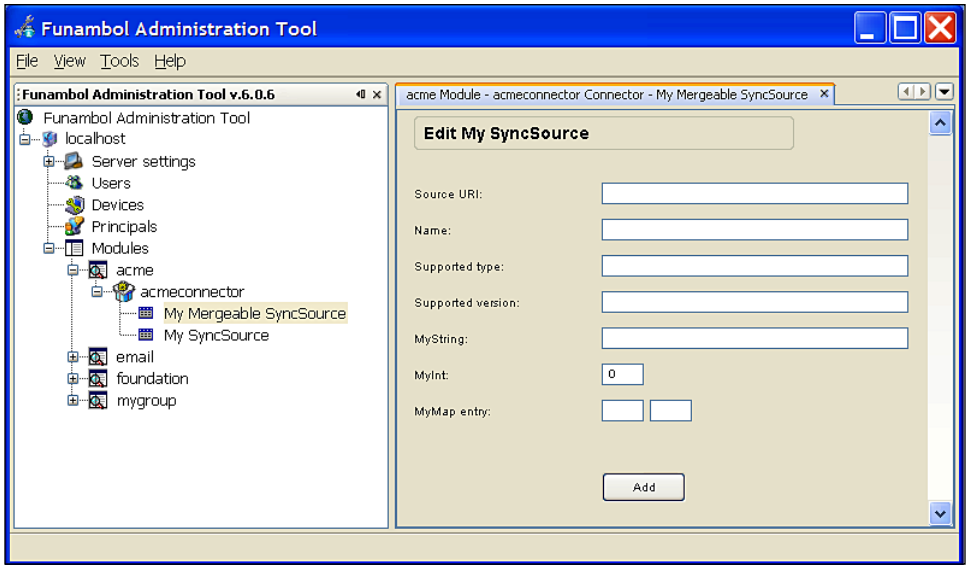
```
[iterate]        (y,n)
y
[echo] acmeconnector-1.0-SNAPSHOT installation
successfully completed
[war] Warning: selected war files include a WEB-
INF/web.xml which will be ignored (please use webxml
attribute to war task)
[echo] Remove output dir

BUILD SUCCESSFUL
Total time: 12 seconds
```

8. Restart the Data Synchronization Service.

The new connector is now installed.

Creating a SyncSource

Now that the `acmeconnector` is installed, it is displayed in the **Funambol Administration Tool**, under the **Modules** section, as shown in the following screenshot:



Once the new SyncSource type is installed on the server, a new instance of the SyncSource can be created and configured so that it is available to the clients. This is done by means of the Funambol Administration Tool. Right-click the `MyMergeableSyncSource` node and select **New** from the context menu that pops up. The form illustrated on the right side of the previous screenshot will appear.

This window is used to specify configuration values. Insert the following values and click **Add**:

```
Source URI: acme
Name: Acme
Supported type: text/plain
Supported version: 1.0
MyString: acme connector!
MyInt: 10
MyMap entry: <acme, connector>
```

Testing the connector

To test the Acme connector, a simple command line client will be used. This is distributed in the Funambol SDK under `$FUNAMBOL_SDK_HOME/plug-ins/cl`.

Before using it, the command line client must be configured to access the SyncSource created earlier. Edit the file `config/spds/sources/briefcase.properties` and set the following values:

```
name=acme
sourceClass=com.funambol.syncclient.spds.source.
FileSystemSyncSource
sourceDirectory=db/briefcase
type=text/plain
sync=two-way
encode=true
sourceURI=acme
```

The command line tool is a simple client that synchronizes the content of a given directory. This is the one specified by the `sourceDirectory` parameter. In the case of Acme connector, however, we are not interested in the data transfer, but more interested in learning the calls executed in the connector SyncSource.

To launch the client, run the command `$FUNAMBOL_SDK_HOME\plug-ins\cl\binrun.cmd` on Windows (or `$FUNAMBOL_SDK_HOME/plug-ins/cl/run.sh` if using Linux).

An output similar to the following will be displayed:

```
Funambol Command Line Tool ver. 8.0.1
--------------------------
2008-04-17 16:28:10:969 - # SyncClient API J2SE Log
16:28:10:970 [INFO] - Initializing
16:28:10:973 [INFO] - Sending initialization commands
16:28:11:716 [INFO] - The server alert code for acme is
201
6:28:11:718 [INFO] - Synchronizing acme
16:28:11:745 [INFO] - exchange modifications started
16:28:11:746 [INFO] - Preparing slow sync for acme
16:28:11:747 [INFO] - Detected 0 items
16:28:11:748 [INFO] - Sending modifications
16:28:11:837 [INFO] - Returned 0 new items, 0 updated
items, 0 deleted items for acme
16:28:11:838 [INFO] - Mapping started
16:28:11:841 [INFO] - Sending mapping
16:28:11:853 [INFO] - Sending mapping
16:28:11:874 [INFO] - Mapping done
16:28:11:874 [INFO] - Synchronization done
```

The server log shows the Acme connector at work. After filtering out the lines that are not of interest for our connector, the log entries will be similar to the following text:

```
[2008-05-17 16:31:56,656] [funambol.myconnector] [INFO]
[399F31F06404433DE69A05F71D562ACD] [sc-api-j2se] [guest]
[] Initializing acme.MyMergeableSyncSource
[2008-05-17 16:31:56,657] [funambol.myconnector] [INFO]
[399F31F06404433DE69A05F71D562ACD] [sc-api-j2se] [guest]
[] myString: acme connector!
[2008-05-17 16:31:56,657] [funambol.myconnector] [INFO]
[399F31F06404433DE69A05F71D562ACD] [sc-api-j2se] [guest]
[] myInt: 10
[2008-05-17 16:31:56,657] [funambol.myconnector] [INFO]
[399F31F06404433DE69A05F71D562ACD] [sc-api-j2se] [guest]
[] myMap: {acme=connector}
[2008-05-17 16:31:56,703] [funambol.myconnector] [INFO]
[399F31F06404433DE69A05F71D562ACD] [sc-api-j2se] [guest]
[acme] Starting synchronization: com.funambol.framework.
engine.source.SyncContext@e85079
[2008-05-17 16:31:56,703] [funambol.myconnector] [INFO]
[399F31F06404433DE69A05F71D562ACD] [sc-api-j2se] [guest]
[acme] getNewSyncItemKeys()
[2008-05-17 16:31:56,703] [funambol.myconnector] [INFO]
[399F31F06404433DE69A05F71D562ACD] [sc-api-j2se] [guest]
[acme] getUpdatedSyncItemKeys()
[2008-05-17 16:31:56,703] [funambol.myconnector] [INFO]
[399F31F06404433DE69A05F71D562ACD] [sc-api-j2se] [guest]
[acme] getDeletedSyncItemKeys()
[2008-05-17 16:31:56,703] [funambol.myconnector] [INFO]
[399F31F06404433DE69A05F71D562ACD] [sc-api-j2se] [guest]
[acme] Committing synchronization
[2008-05-17 16:31:56,782] [funambol.myconnector] [INFO]
[399F31F06404433DE69A05F71D562ACD] [sc-api-j2se] [guest]
[acme] Ending synchronization
```

Note: The first entries of the log show the values inserted earlier in the administration panel, which means that the SyncSource configuration was properly picked up.

Similarly, the server log also shows the effect of `MySynclet`. Each SyncML message has been logged; for example the first input message is something like the following text:

```
[2008-05-17 16:59:28,576] [funambol.myconnector] [INFO]
[FF4B335BA02F0581DAFF016319EDD263] [] [] []
[2008-05-17 16:59:28,576] [funambol.myconnector] [INFO]
[FF4B335BA02F0581DAFF016319EDD263] [] [] []
Input message[2008-05-17 16:59:28,576] [funambol.
myconnector] [INFO] [FF4B335BA02F0581DAFF016319EDD263] []
[] []                              [2008-05-17 16:59:28,585]
[funambol.myconnector] [INFO] [FF4B335BA02F0581DAFF016319
EDD263] [] [] []
<?xml version="1.0" encoding="UTF-8"?>
<SyncML><SyncHdr><VerDTD>1.1</VerDTD><VerProto>
SyncML/1.1</VerProto><SessionID>12345678</
SessionID><MsgID>1</MsgID><Target><LocURI>http://
localhost:8080/funambol/ds</LocURI></Target>
<Source><LocURI>sc-api-j2se</LocURI></Source>
<Cred><Meta><Type>syncml:auth-basic</Type></Meta>
<Data>Z3Vlc3Q6Z3Vlc3Q=</Data></Cred><Meta><MaxMsgSize>25
0000
</MaxMsgSize><MaxObjSize>4000000</MaxObjSize></Meta></
SyncHdr>
<SyncBody><Alert><CmdID>1</CmdID><Data>200</
Data><Item><Target>
<LocURI>acme</LocURI></Target><Source><LocURI>acme</
LocURI></Source>
<Meta><Anchor><Last>1211034716596</Last><Next>1211036368
401</Next>
</Anchor></Meta></Item></Alert><Final/></SyncBody></
SyncML>[2008-05-17 16:59:28,585] [funambol.myconnector]
[INFO] [FF4B335BA02F0581DAFF016319EDD263] [] [] [] ------
----------------------------------------------------------
```

The last output message is something like the following:

```
[2008-05-17 16:59:28,876] [funambol.myconnector] [INFO]
[FF4B335BA02F0581DAFF016319EDD263] [sc-api-j2se] [guest]
[]
[2008-05-17 16:59:28,877] [funambol.myconnector] [INFO]
[FF4B335BA02F0581DAFF016319EDD263] [sc-api-j2se] [guest]
[]
Output message[2008-05-17 16:59:28,877] [funambol.
myconnector] [INFO] [FF4B335BA02F0581DAFF016319EDD263]
[sc-api-j2se] [guest] []
[2008-05-17 16:59:28,877] [funambol.myconnector] [INFO]
[FF4B335BA02F0581DAFF016319EDD263] [sc-api-j2se] [guest]
[]
<?xml version="1.0" encoding="UTF-8"?>
<SyncML><SyncHdr><VerDTD>1.1</VerDTD><VerProto>SyncML/1.1
</VerProto>
<SessionID>12345678</SessionID><MsgID>5</
MsgID><Target><LocURI>sc-api-j2se</LocURI></
Target><Source><LocURI>
http://localhost:8080/funambol/ds</LocURI></Source>
<RespURI>http://localhost:8080/funambol/ds;jsessionid=FF4
B335BA02F0581DAFF016319EDD263</RespURI></SyncHdr><SyncBod
y><Status><CmdID>1
</CmdID><MsgRef>5</MsgRef><CmdRef>0</
CmdRef><Cmd>SyncHdr</Cmd><TargetRef>http://
localhost:8080/funambol/ds</TargetRef><SourceRef>sc-
api-j2se</SourceRef><Data>200</Data></Status><Final/></
SyncBody></SyncML>[2008-05-17 16:59:28,877] [funambol.
myconnector] [INFO] [FF4B335BA02F0581DAFF016319EDD263]
[sc-api-j2se] [guest] [] ------------------------------
----------------------------
```

If you have reached this point, you have successfully developed and deployed a Funambol connector. Congratulations! You can now further extend the `acmeconnector` project or start your own.

Debugging

When developing a connector, it can be very helpful to to use a debugger to inspect the behavior of the connector to discover and fix bugs. The best way to do this is to use the remote debugging capabilities provided by the **Java Virtual Machine** (**JVM**). To do so, follow these steps:

1. Edit the file `$FUNAMBOL_HOME/bin/funambol-server` uncommenting the `JPDA_OPTS` line, which you can easily find if you search for "debug". This will start Funambol, enabling it for remote debugging.
2. Connect to the running JVM. If you use Eclipse, you can create a debug profile to attach to Funambol. Note that the debug port given in the `JPDA_OPTS` line is 8787, not eclipse's default port, (8000), so either change the `JDPA_OPTS` or the Eclipse port. NetBeans supports remote debugging too. Click on the **Debug** menu item and then select **Attach Debugger**.
3. Now you can start a sync session and stop at breakpoints.

Summary

This chapter helps developers who want to extend the Funambol DS Service to integrate with their own backend datasources. This chapter presented some key concepts of the synchronization process, which provided us with the background information needed to understand some of the details of the Funambol API and framework. It also showed that developing a connector can be pretty straightforward. By following the steps presented in this chapter, a new simple connector can be created and deployed. This is only a starting point, but it allows developers to see the overall synchronization flow, from a client making a synchronization request, to the connector code.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Thank you for buying

Funambol Mobile Open Source

Packt Open Source Project Royalties

When we sell a book written on an Open Source project, we pay a royalty directly to that project. Therefore by purchasing Funambol Mobile Open Source, Packt will have given some of the money received to the Funambol project.

In the long term, we see ourselves and you—customers and readers of our books—as part of the Open Source ecosystem, providing sustainable revenue for the projects we publish on. Our aim at Packt is to establish publishing royalties as an essential part of the service and support a business model that sustains Open Source.

If you're working with an Open Source project that you would like us to publish on, and subsequently pay royalties to, please get in touch with us.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

About Packt Publishing

Packt, pronounced 'packed', published its first book "Mastering phpMyAdmin for Effective MySQL Management" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.PacktPub.com`.